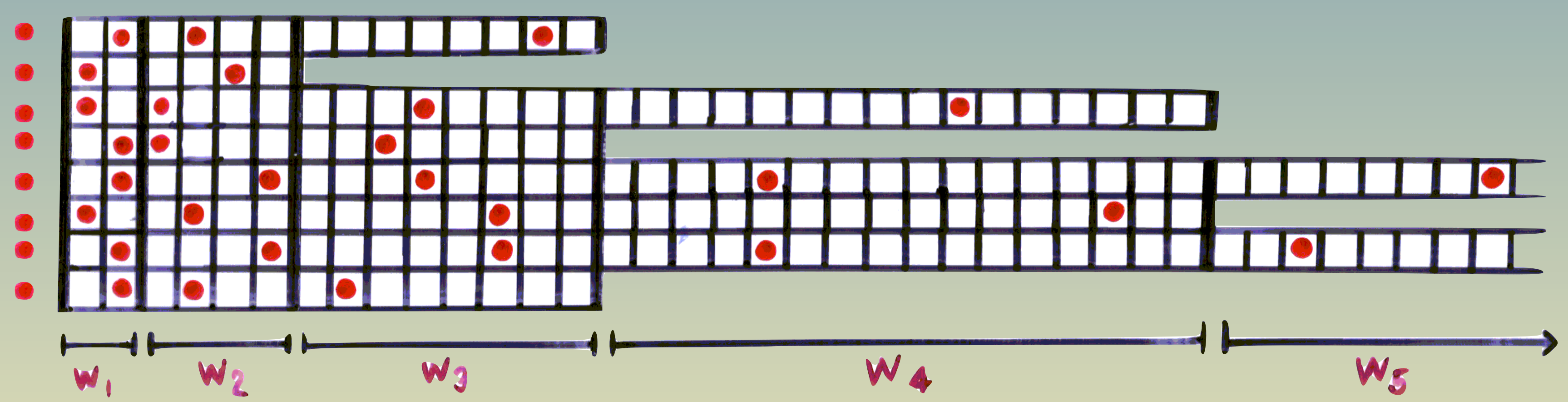
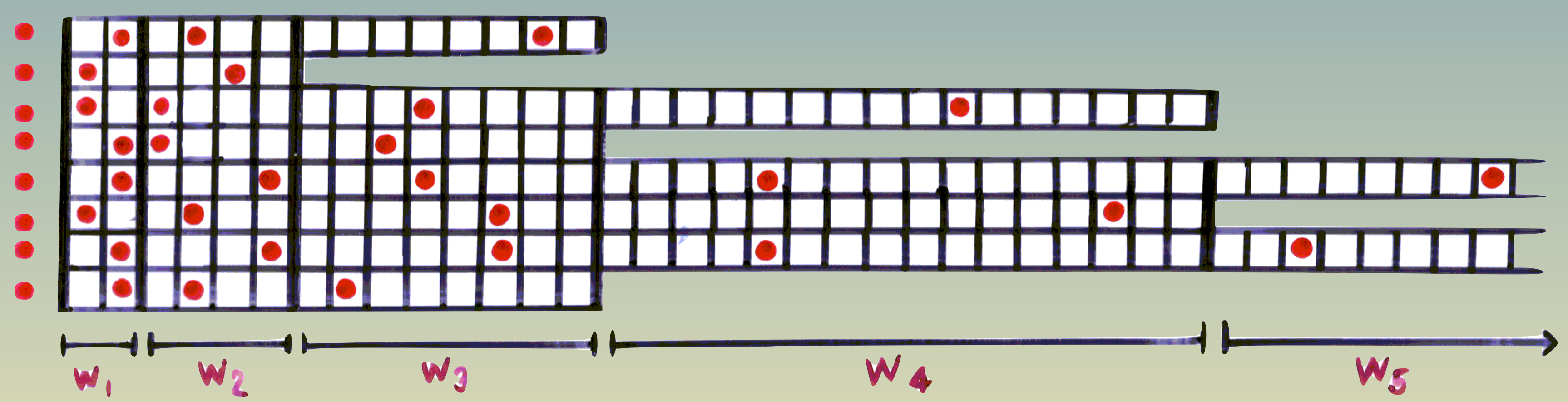


W_1
W_2
W_3
W_4
W_5

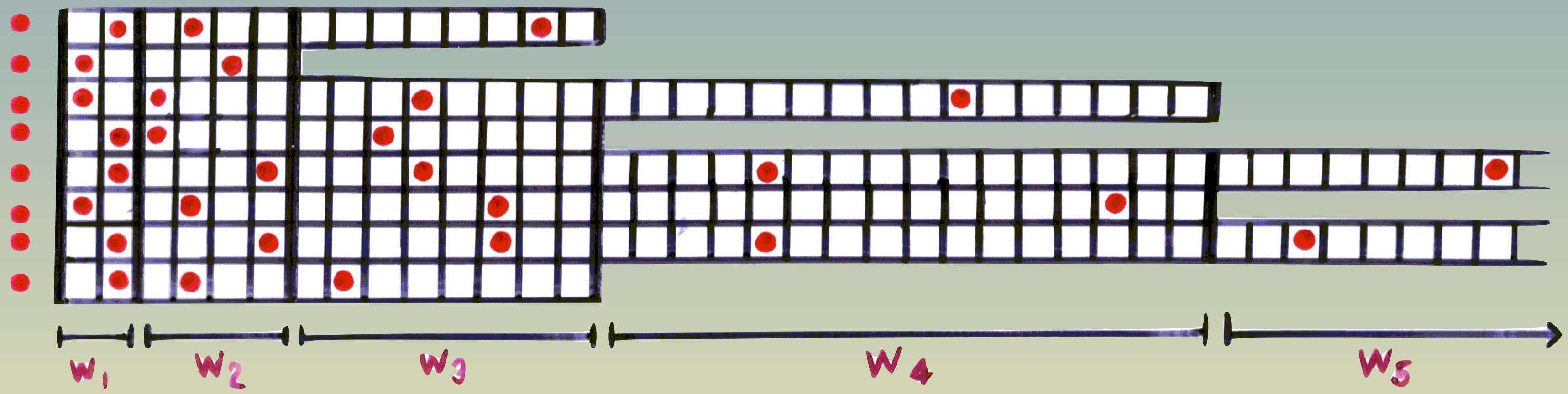
TBD



TBD

Three backoff dilemmas

Michael A. Bender



Stony Brook University

Tokutek

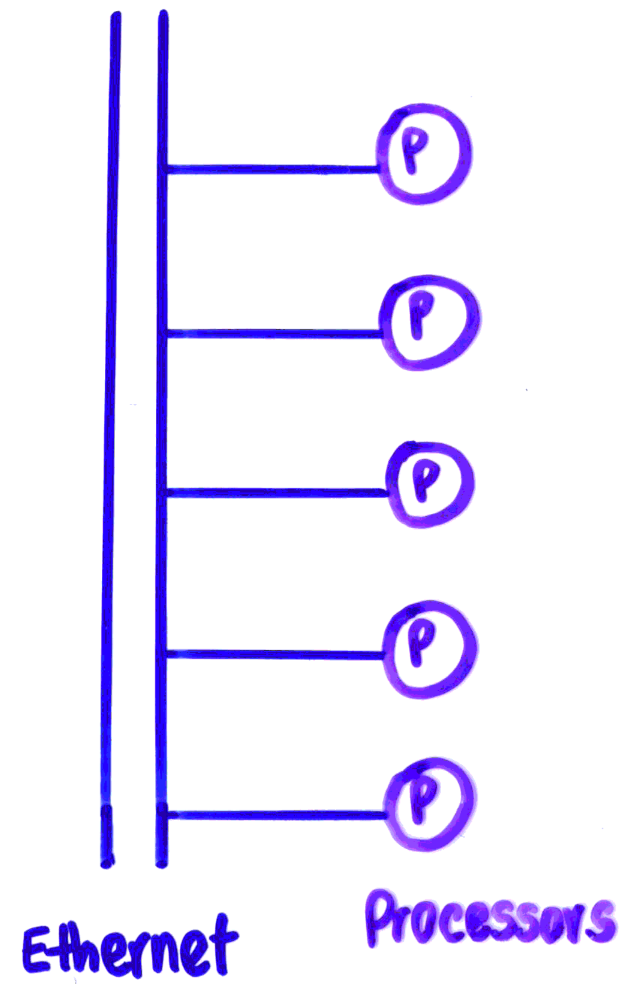
Backoff is about sharing

Classic scenario:

- Many devices.
- 1 (shared) resource.
- Only one device can access the resource at a time!

Examples:

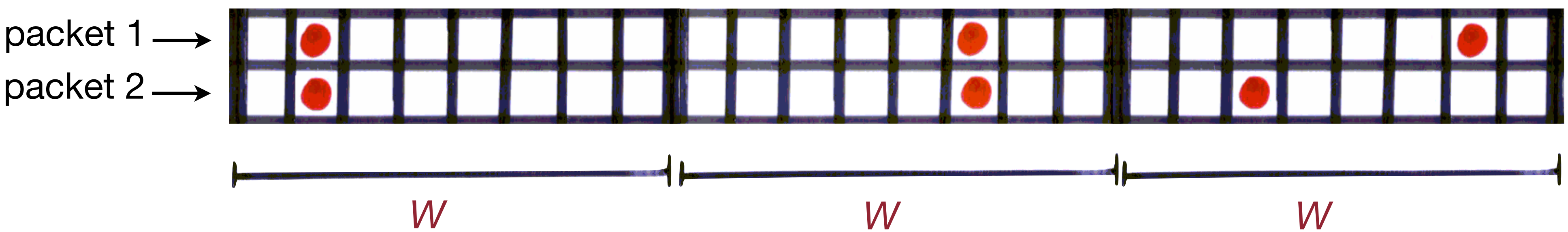
- LANs
- Wireless networks
- Transactional memory
- Lock acquisition
- E-mail retransmission
- Congestion control (e.g., TCP)



Randomized backoff [Abramson '70]

Repeat until resource acquired

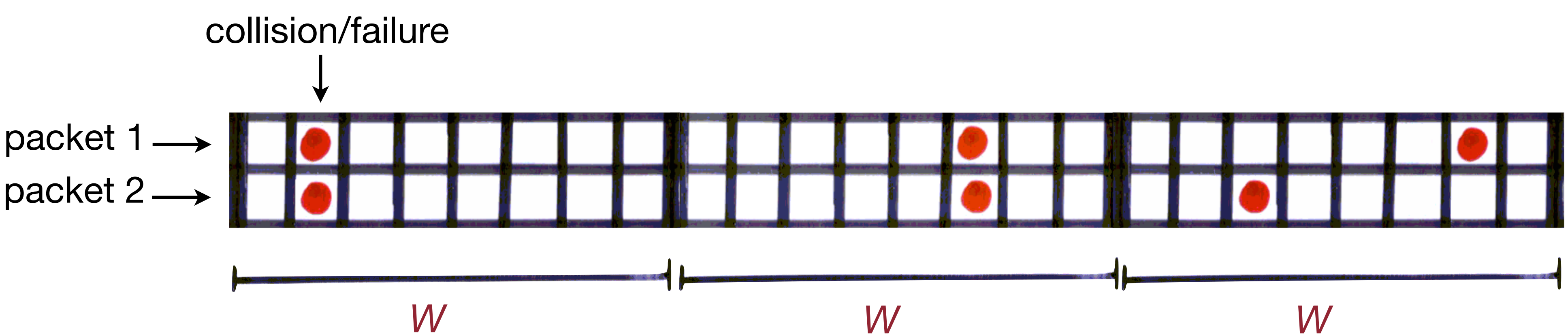
- **Try to grab resource**
- **If failed then**
  randomly choose *t* in window [1,10]
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until resource acquired

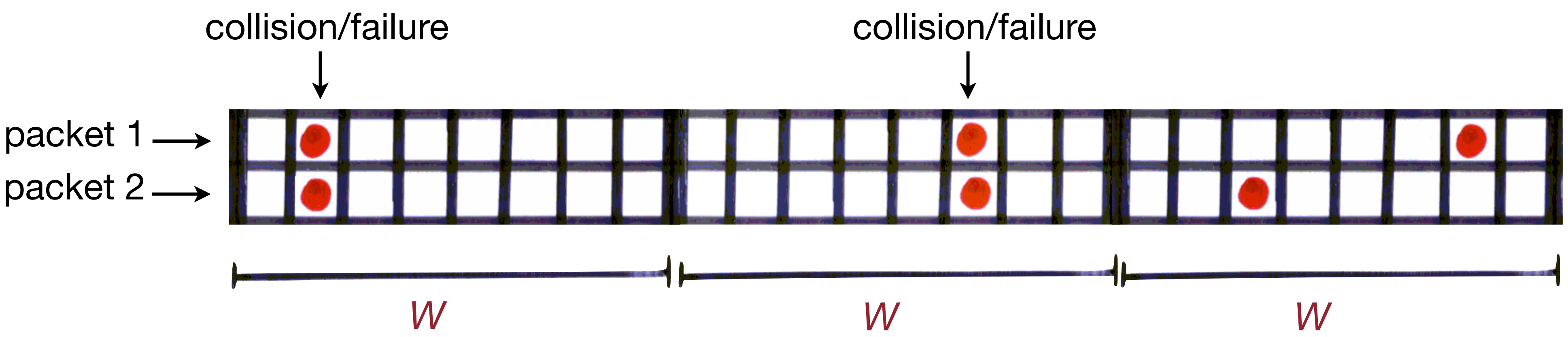
- **Try to grab resource**
- **If failed then**
  randomly choose *t* in window [1,10]
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until resource acquired

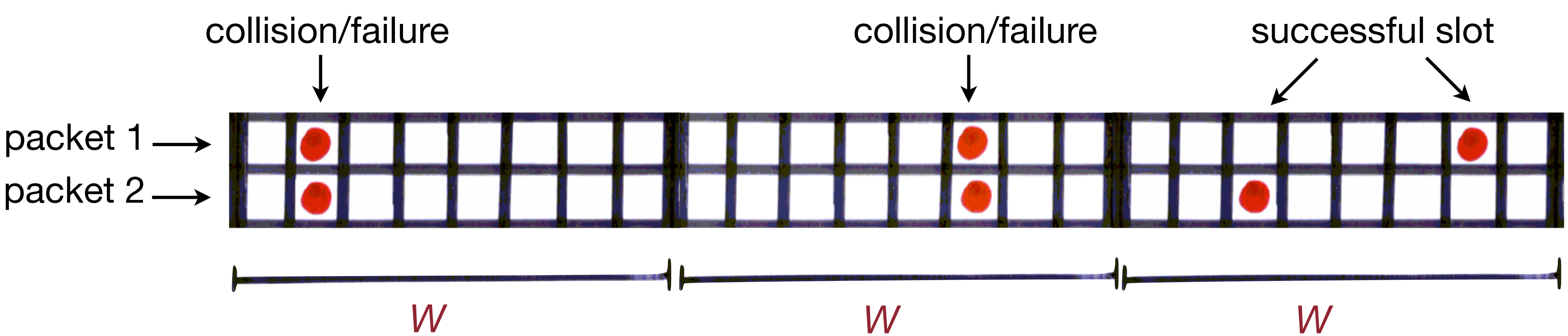
- **Try to grab resource**
- **If failed then**
  randomly choose t in window [1,10]
  and wait t seconds.



Randomized backoff [Abramson '70]

Repeat until resource acquired

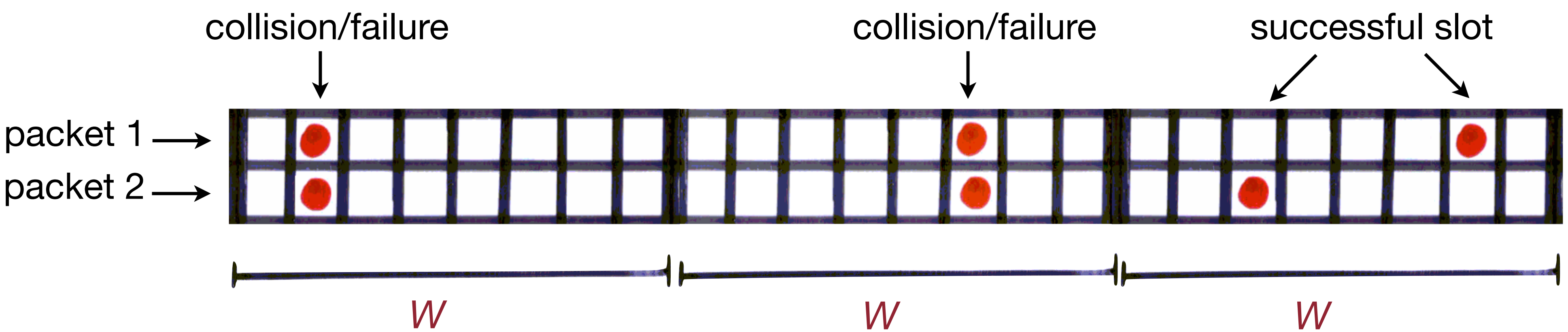
- **Try to grab resource**
- **If failed then**
  randomly choose *t* in window [1,10]
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until resource acquired

- **Try to grab resource**
- **If failed then**
  randomly choose *t* in window [1,10]
  and wait *t* seconds.

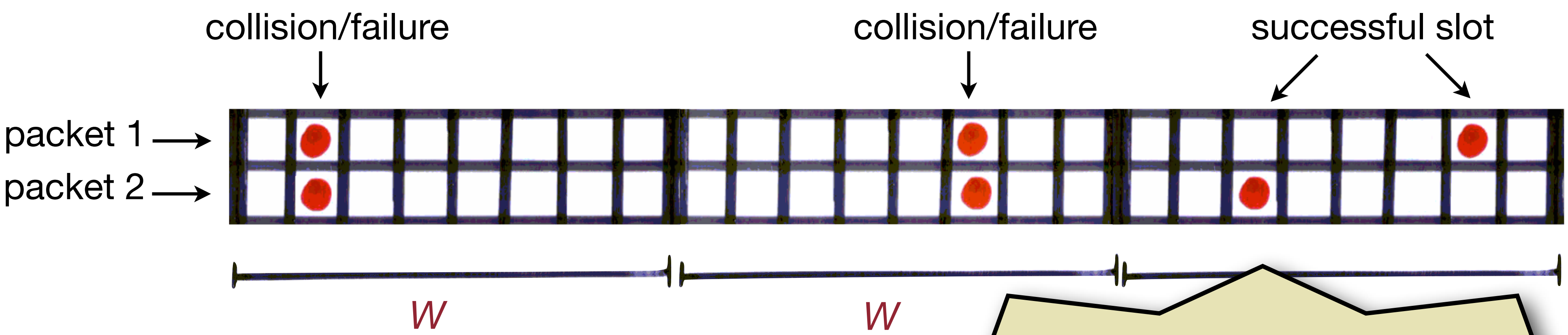


Bad scenario: thousands of devices contend for the resource.

Randomized backoff [Abramson '70]

Repeat until resource acquired

- **Try to grab resource**
- **If failed then**
  randomly choose *t* in window [1,10]
  and wait *t* seconds.



Basic backoff question:
How to choose and adapt the window size *W*.

Bad scenario: thousands of devices contend for the resource.

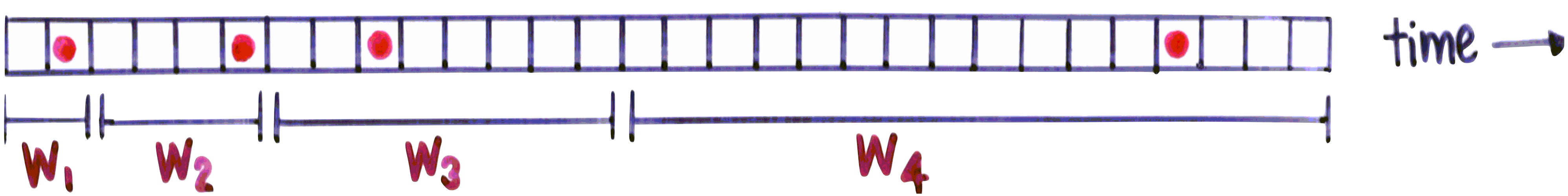
Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

Window size $W = 2$

Repeat until resource acquired:

- **Randomly choose slot t in window.**
- **Try to grab resource at slot t.**
- **If failed, wait to end of W. Then double W.**



Standard answer: Binary exponential backoff

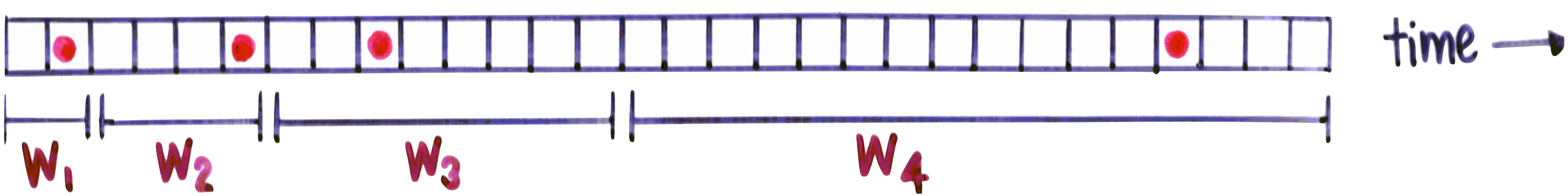
[Metcalfe and Boggs '76]

Window size *W* = 2

Repeat until resource acquired:

- Randomly choose slot *t* in window.
- Try to grab resource at slot *t*.
- If failed, wait to end of *W*.
  Then double *W*.

Why double?
What if the window size changes by a different factor?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

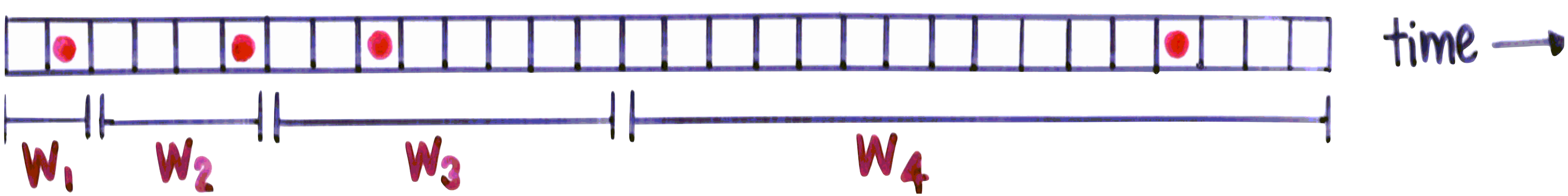
Window size W = 2

How many attempts to acquire the resource until a success?

Repeat until resource acquired:

- **Randomly choose slot t in window.**
- **Try to grab resource at slot t.**
- **If failed, wait to end of W.**
  Then double W.

Why double?
What if the window size changes by a different factor?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

Window size *W* = 2

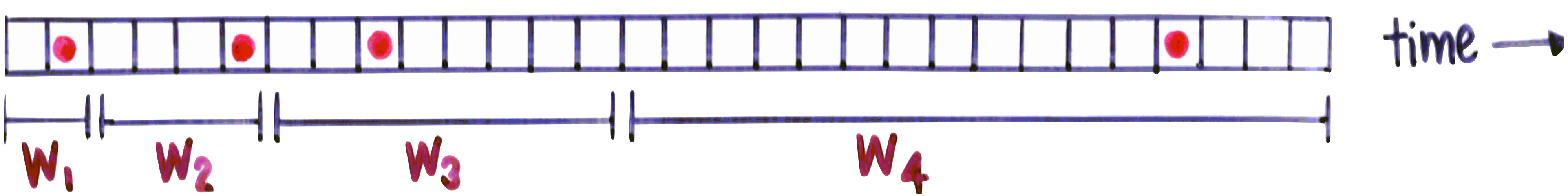
How many attempts to acquire the resource until a success?

Repeat until resource acquired:

- Randomly choose slot *t* in window.
- Try to grab resource at slot *t*.
- If failed, wait to end of *W*. Then double *W*.

How well does exponential backoff deal with bursty arrivals?

Why double?
What if the window size changes by a different factor?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

How many attempts to acquire the resource until a success?

Window size $W = 2$

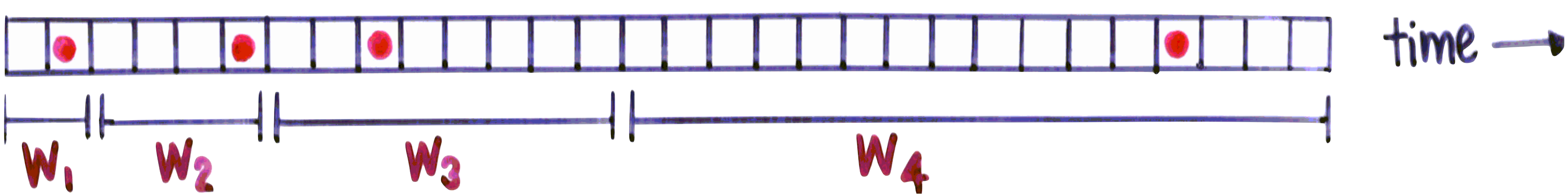
Repeat until resource acquired:

- Randomly choose slot t in window.
- Try to grab resource at slot t.
- If failed, wait to end of W. Then double W.

How well does exponential backoff deal with bursty arrivals?

Are there any throughput guarantees?

Why double? What if the window size changes by a different factor?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

Window size $W = 2$

Repeat until resource acquired:

- **Randomly choose slot t in window.**
- **Try to grab resource at slot t.**
- **If failed, wait to end of W. Then double W.**

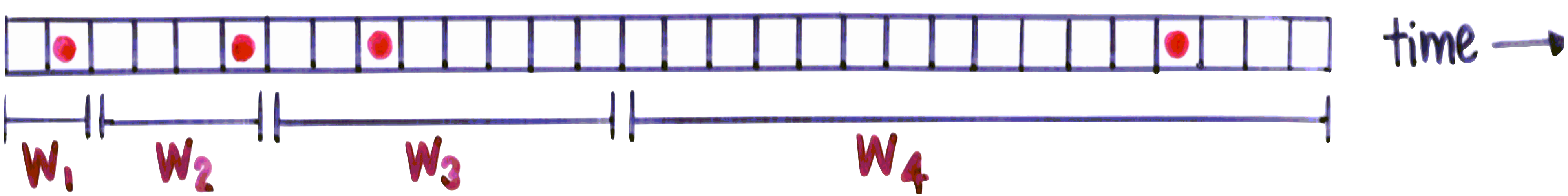
How many attempts to acquire the resource until a success?

How well does exponential backoff deal with bursty arrivals?

Are there any throughput guarantees?

What about robustness guarantees?

Why double? What if the window size changes by a different factor?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

How many attempts to acquire the resource until a success?

Window size $W = 2$

Repeat until resource acquired:

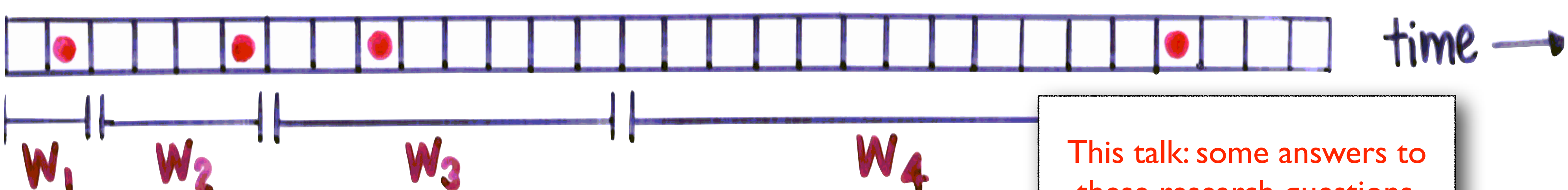
- **Randomly choose slot t in window.**
- **Try to grab resource at slot t.**
- **If failed, wait to end of W. Then double W.**

How well does exponential backoff deal with bursty arrivals?

Are there any throughput guarantees?

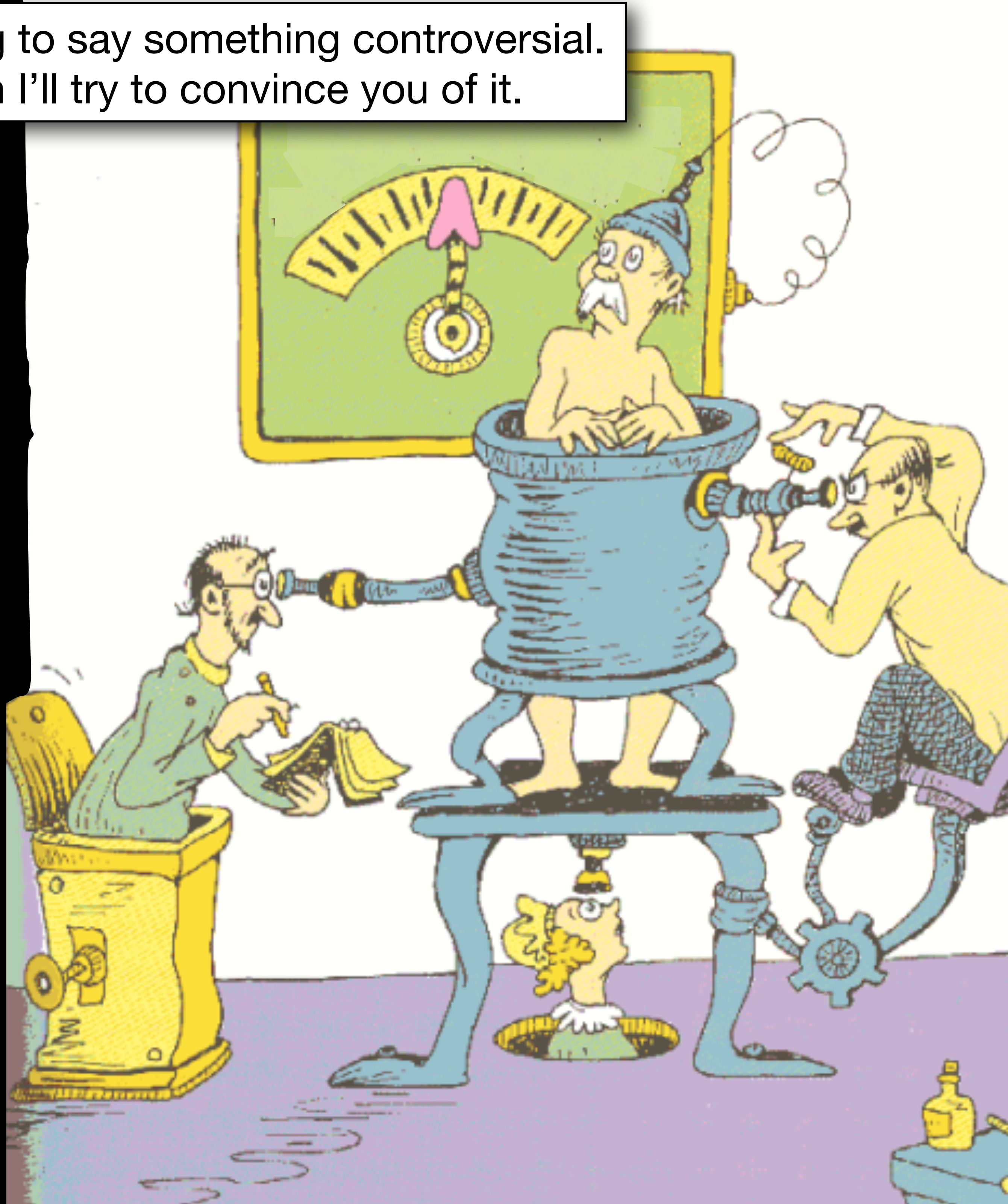
Why double?
What if the window size changes by a different factor?

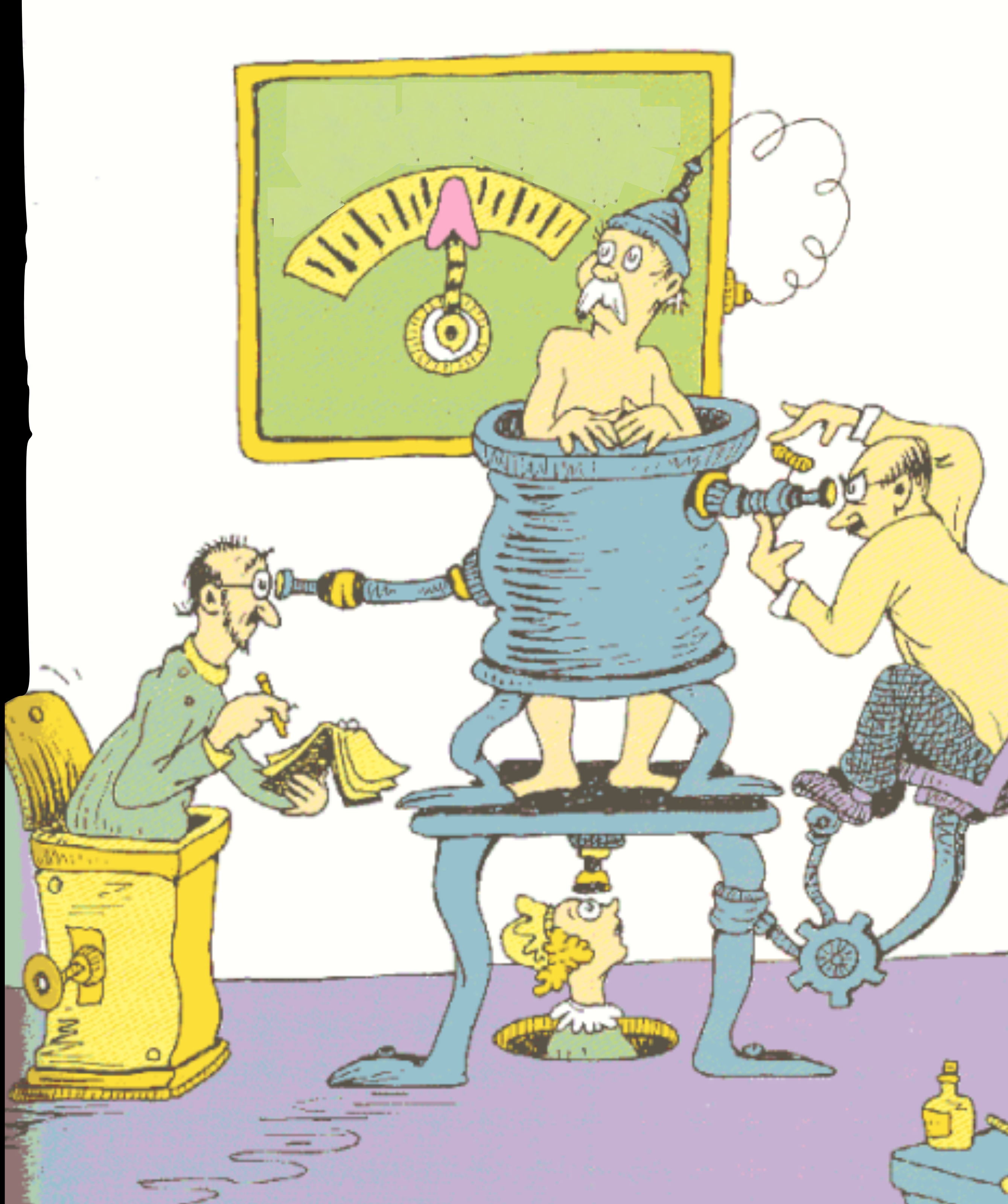
What about robustness guarantees?



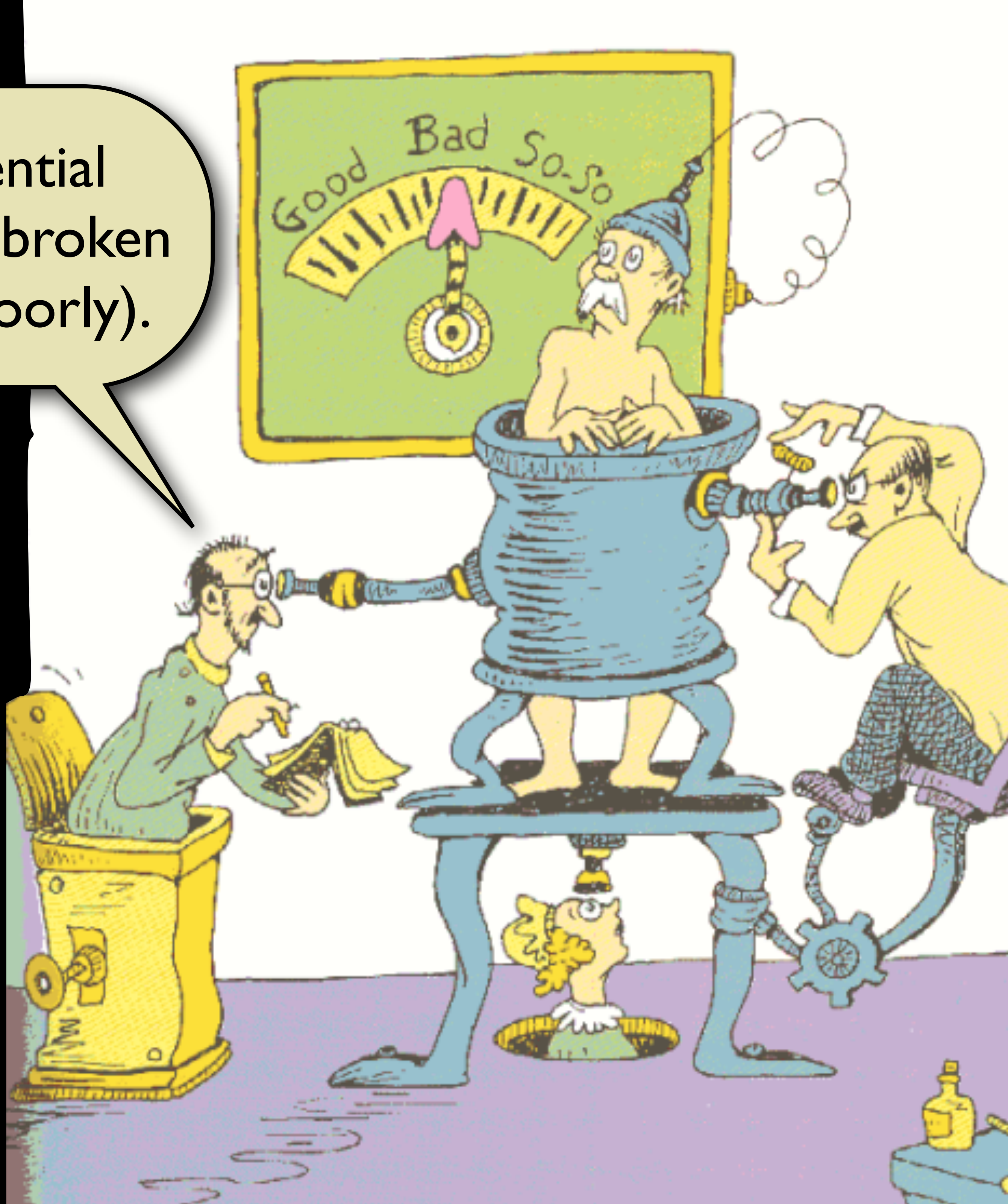
This talk: some answers to these research questions.

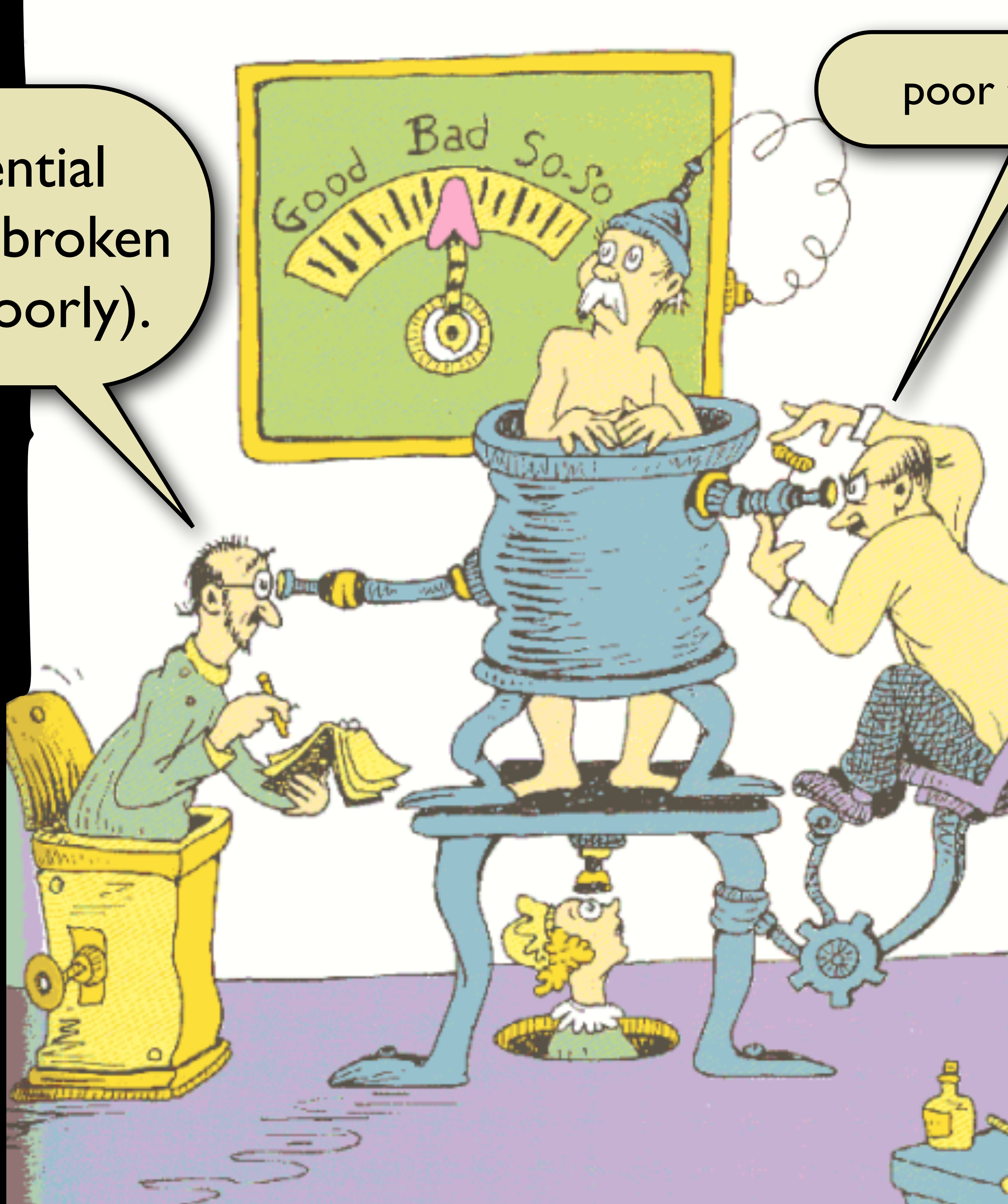
I’m going to say something controversial.
Then I’ll try to convince you of it.

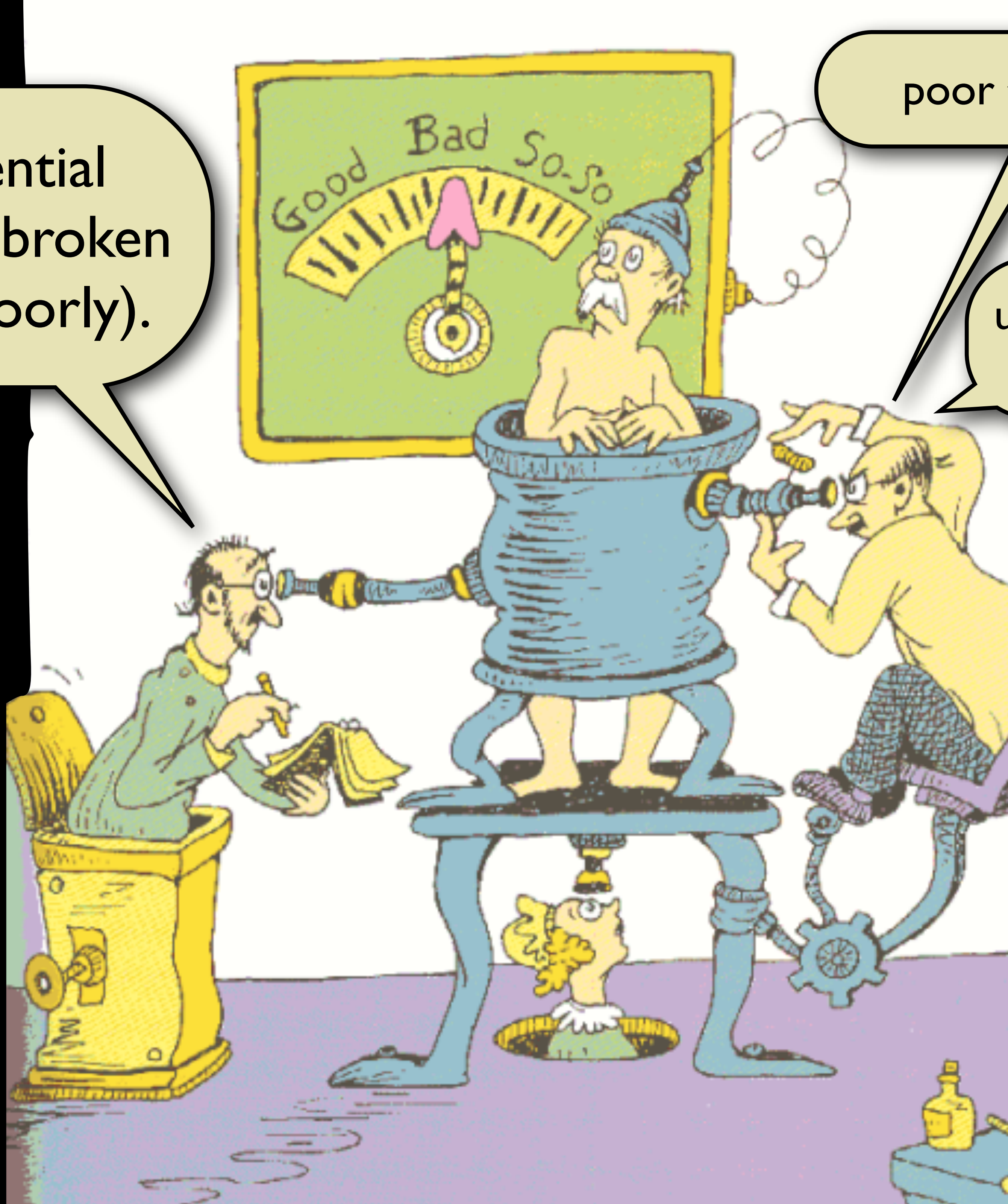


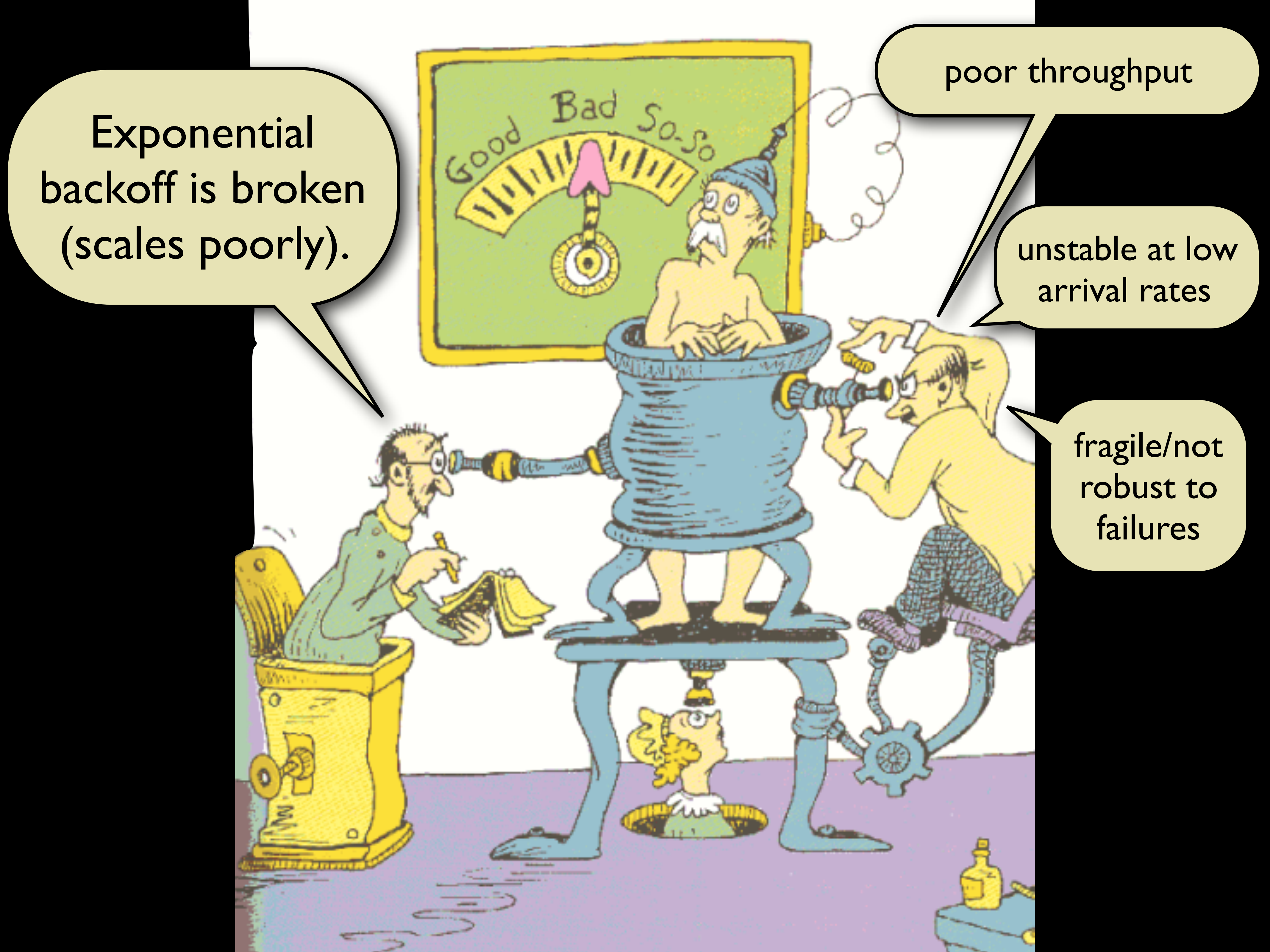


Exponential backoff is broken (scales poorly).

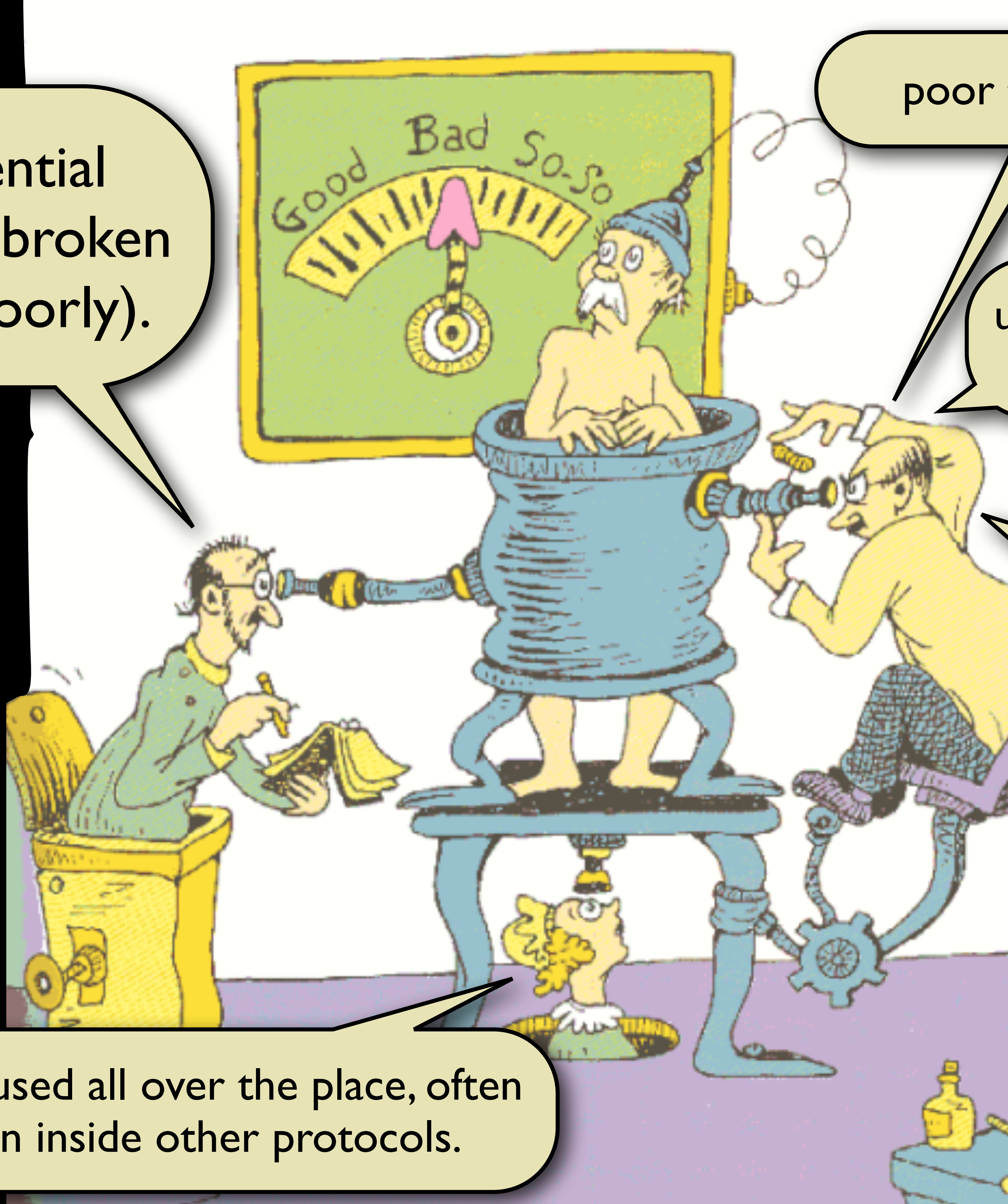



Exponential backoff is broken (scales poorly).
poor throughput
Good
Bad
So-So


Exponential backoff is broken (scales poorly).
poor throughput
unstable at low arrival rates
Good
Bad
So-So



Exponential backoff is broken (scales poorly).
Good
Bad
So-So
poor throughput
unstable at low arrival rates
fragile/not robust to failures



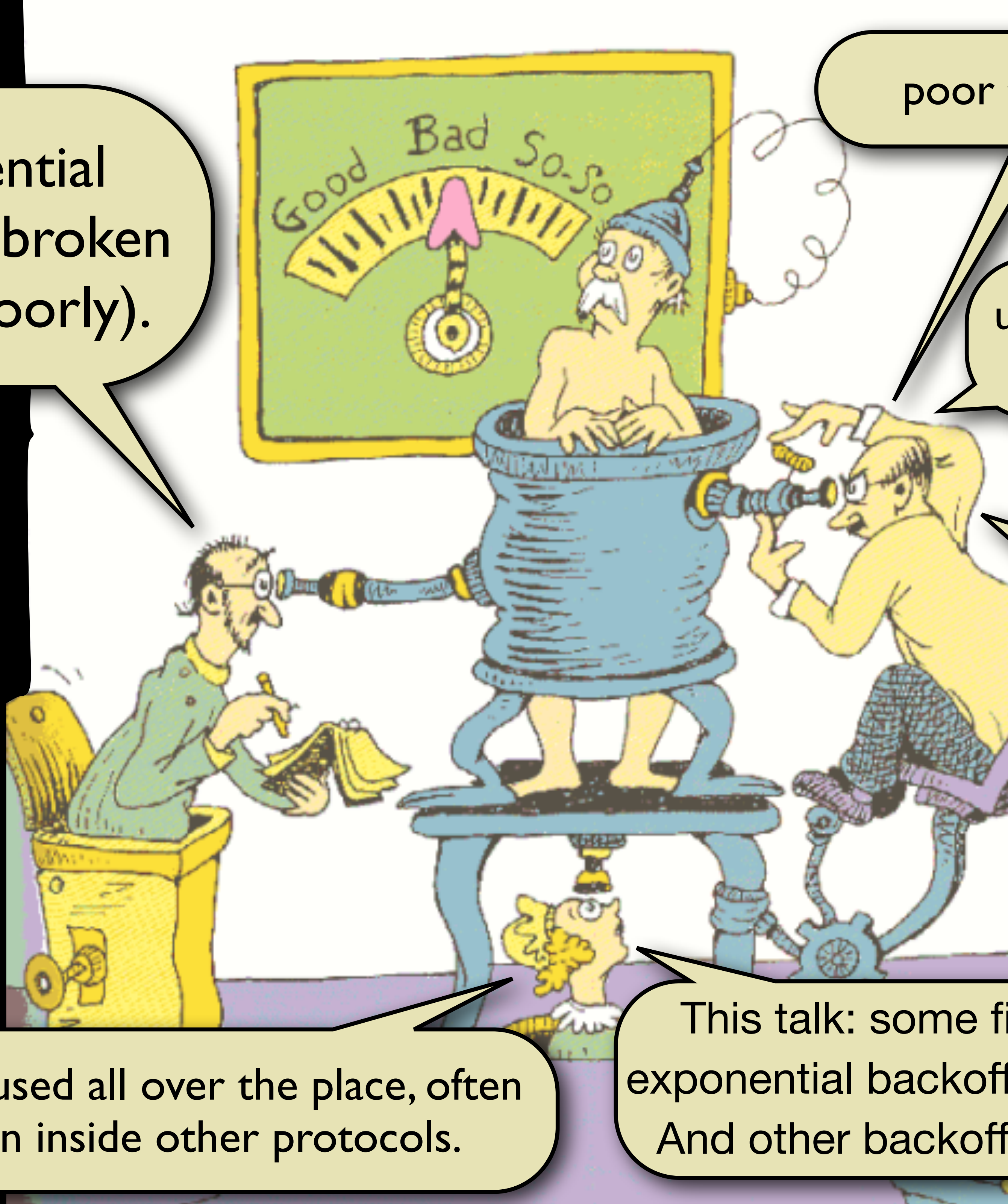
Exponential backoff is broken (scales poorly).

poor throughput

unstable at low arrival rates

fragile/not robust to failures

But it is used all over the place, often hidden inside other protocols.


Good
Bad
So-So
Exponential backoff is broken (scales poorly).
poor throughput
unstable at low arrival rates
fragile/not robust to failures
But it is used all over the place, often hidden inside other protocols.
This talk: some fixes to exponential backoff.
And other backoff algorithms.

This talk

Act 1: Binary exponential backoff is broken.

- batch (single burst)
- dynamic arrivals

Act 2: TBD (three backoff dilemmas).

- how to maximize throughput,
- minimize # tries to access resource, and
- achieve robustness.

Act 3: How to fix exponential backoff.

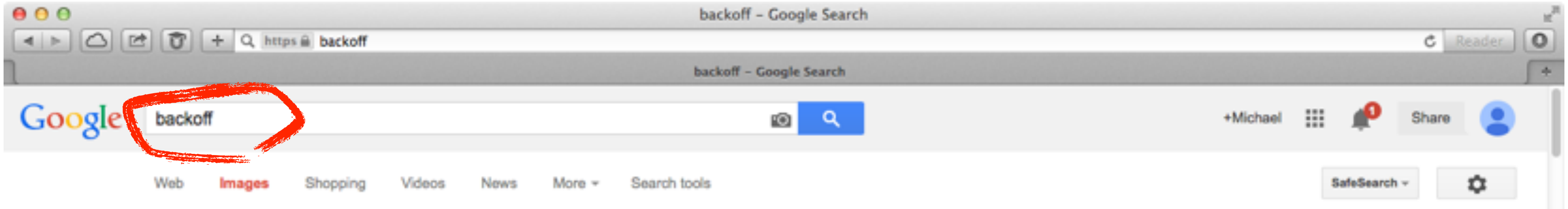
- batch
- dynamic arrivals

Good pictures help convey intuition.

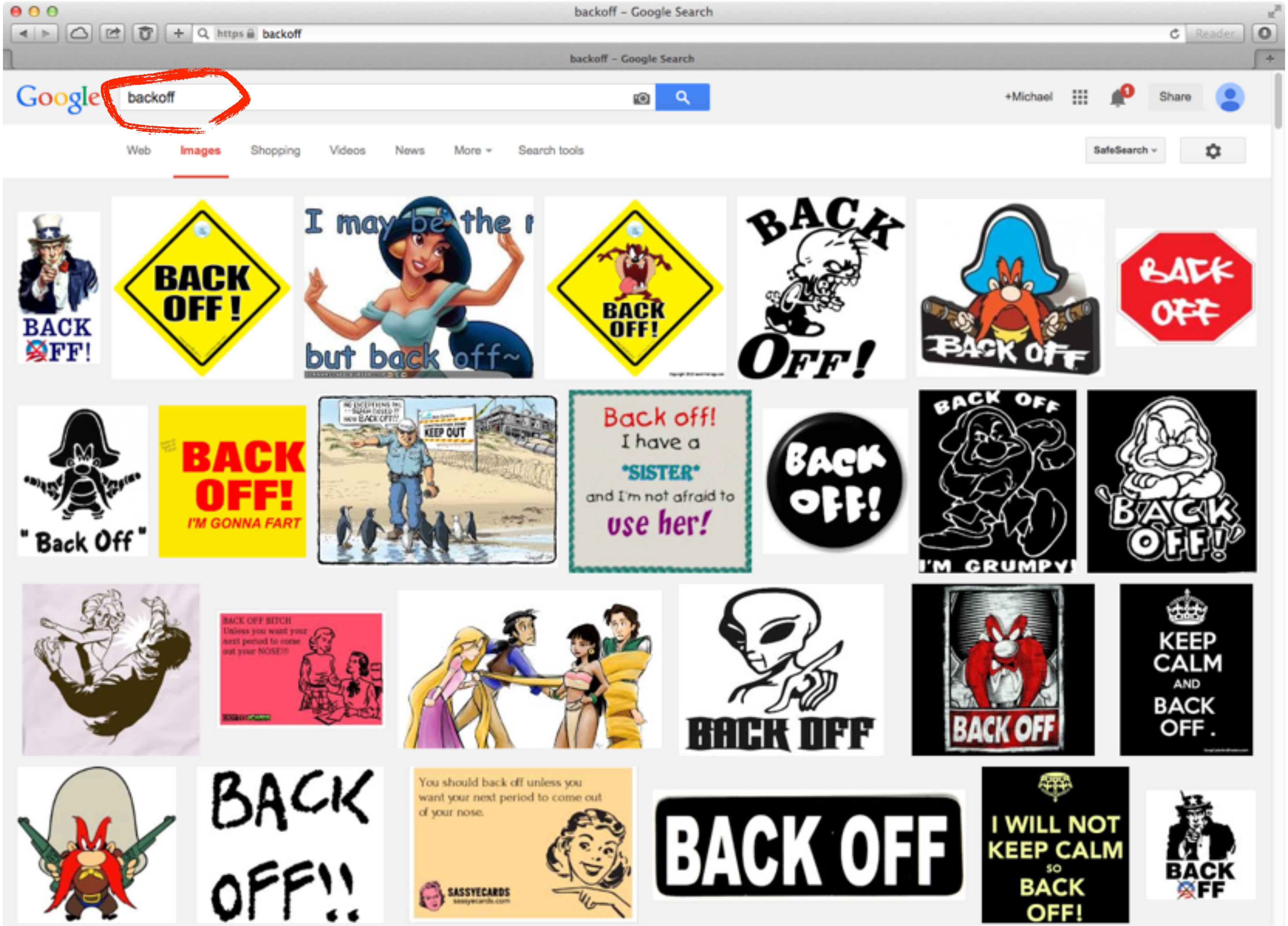
Good pictures help convey intuition.

So in preparing this talk, the first thing I did is type “backoff” into Google.

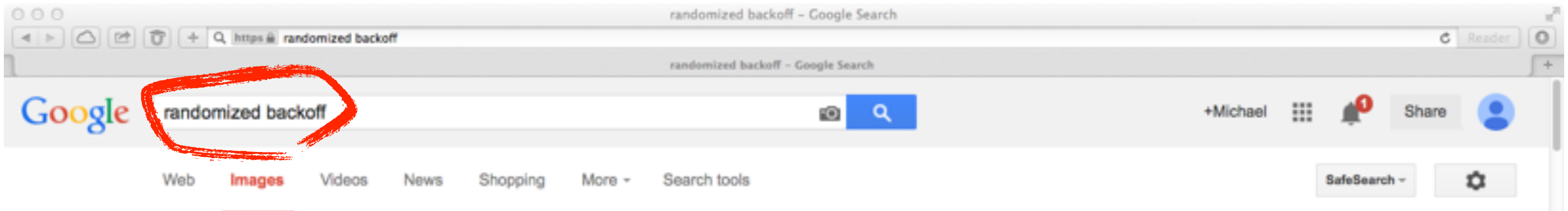
What Google says about backoff is intuitive but off topic.



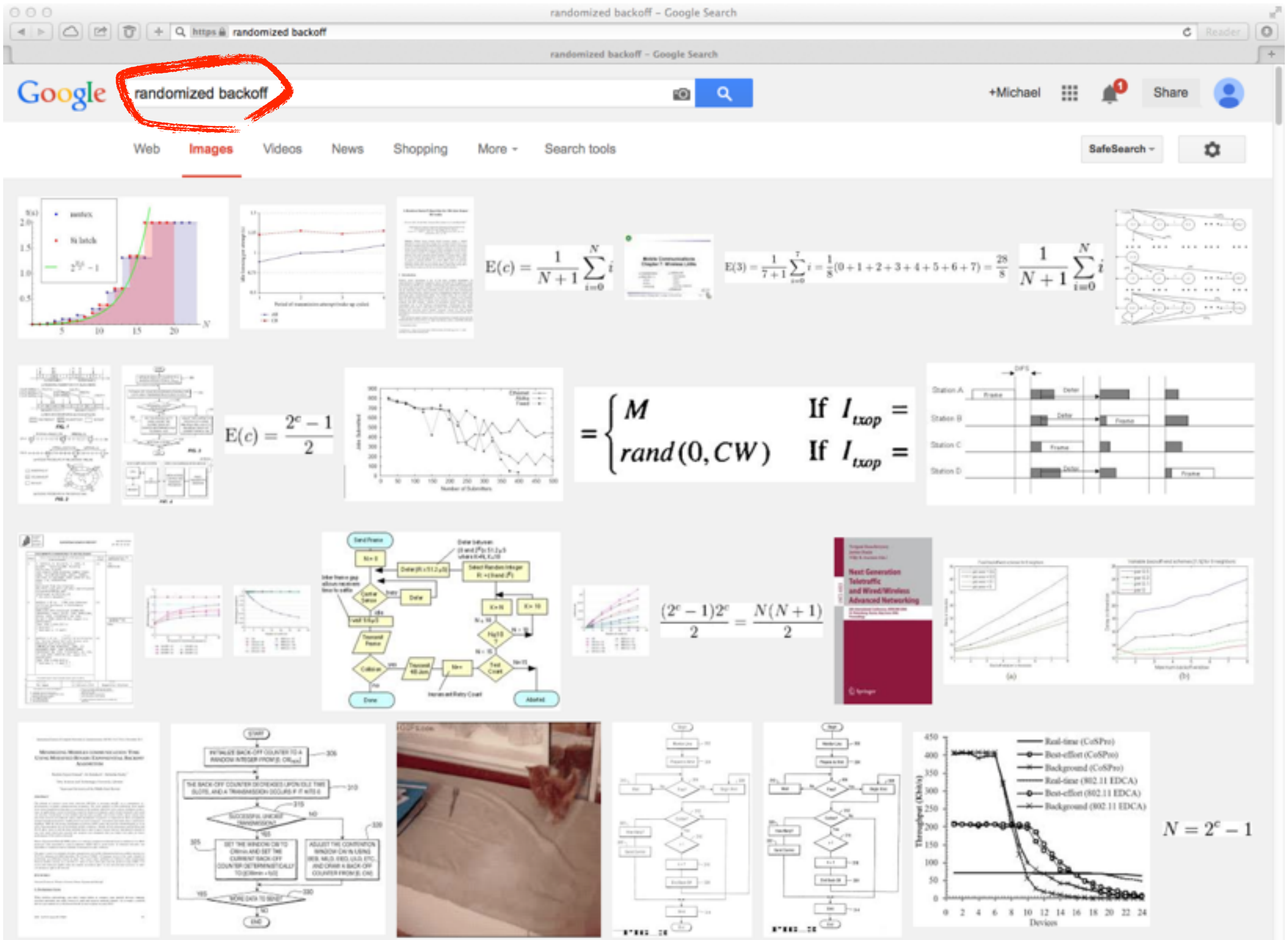
What Google says about backoff is intuitive but off topic.



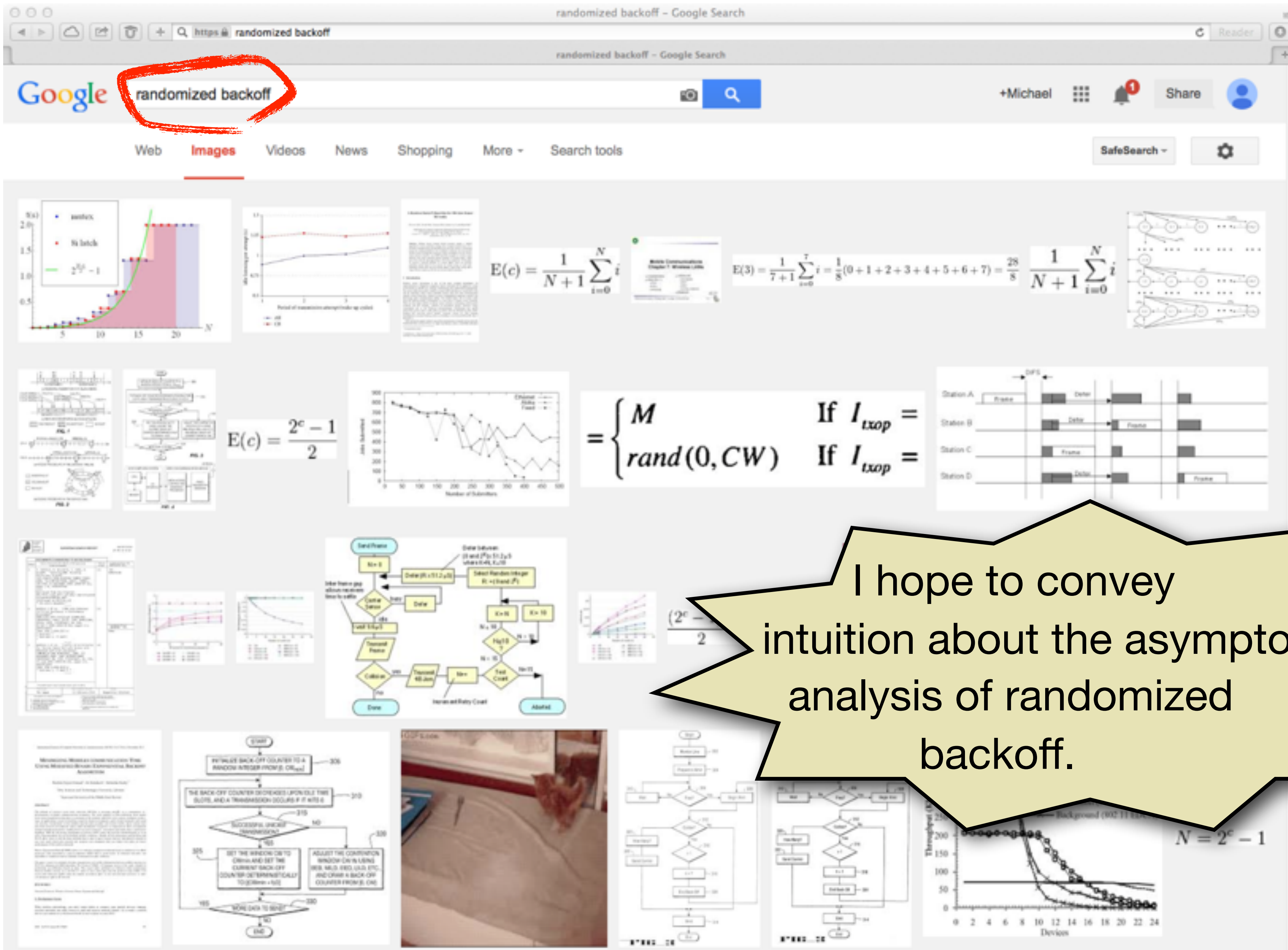
What Google says about "randomized backoff" is on topic but less algorithmic...



What Google says about "randomized backoff" is on topic but less algorithmic...



What Google says about "randomized backoff" is on topic but less algorithmic...



I hope to convey intuition about the asymptotic analysis of randomized backoff.

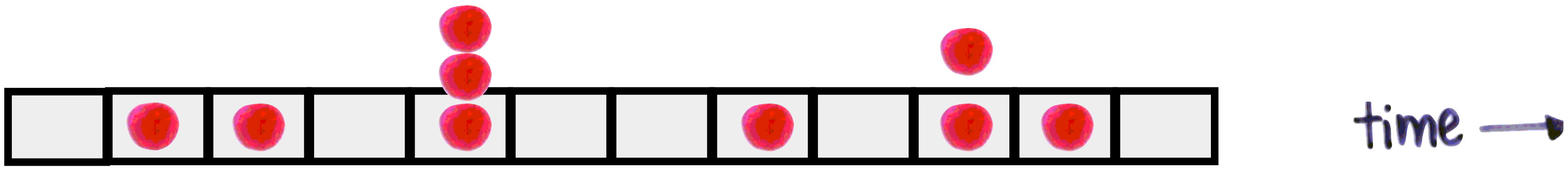
Part 1: binary exponential backoff is broken

Analysis in two settings:

- batch (a single burst)
- dynamic arrivals

Backoff model: multiple-access channel

Time is divided into discrete slots.

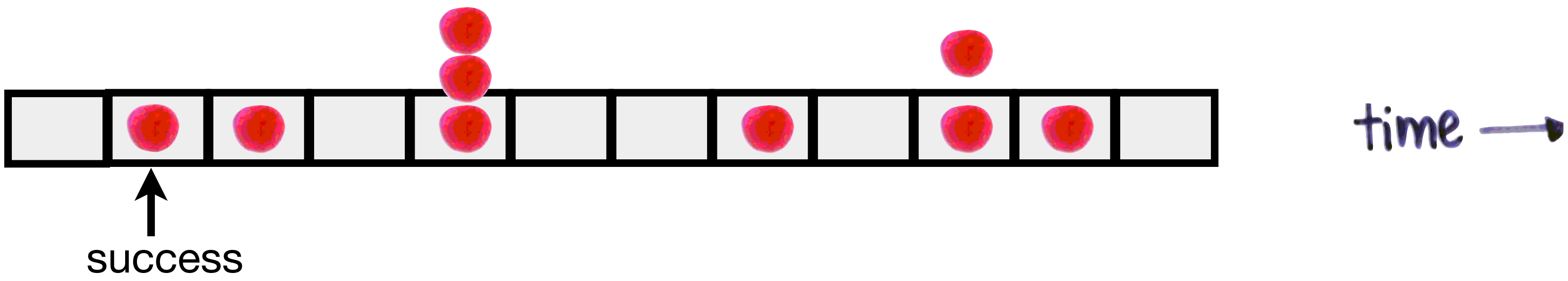


In every slot, a device can:

- **Broadcast** (access the channel)
- **Listen** (sense the channel)

Backoff model: multiple-access channel

Time is divided into discrete slots.



In every slot, a device can:

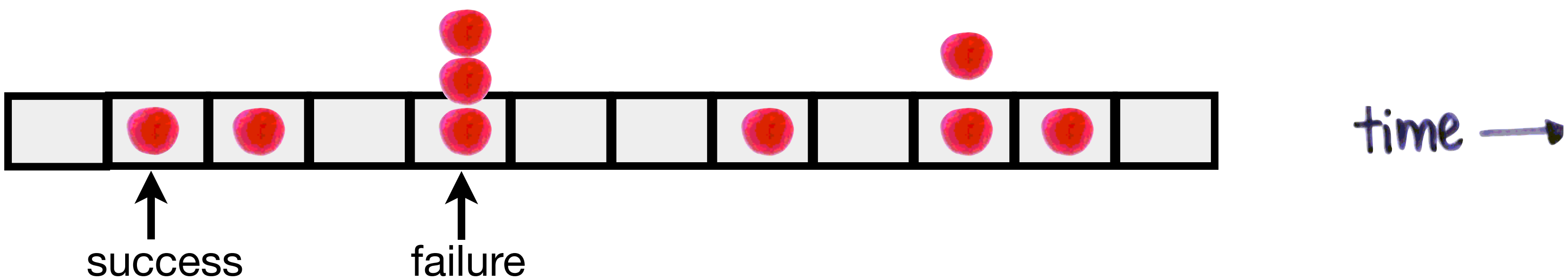
- **Broadcast** (access the channel)
- **Listen** (sense the channel)

Results (known to every broadcaster/listener):

- If **exactly one** device broadcasts, then **success**.
- If **two or more** devices broadcast, then **failure**.
- If **zero** devices broadcast, then **nothing**.

Backoff model: multiple-access channel

Time is divided into discrete slots.



In every slot, a device can:

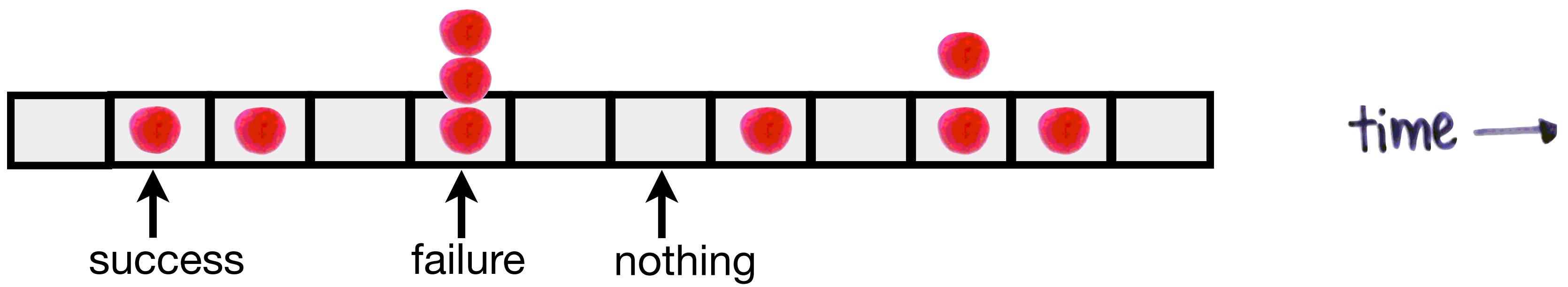
- **Broadcast** (access the channel)
- **Listen** (sense the channel)

Results (known to every broadcaster/listener):

- If **exactly one** device broadcasts, then **success**.
- If **two or more** devices broadcast, then **failure**.
- If **zero** devices broadcast, then **nothing**.

Backoff model: multiple-access channel

Time is divided into discrete slots.



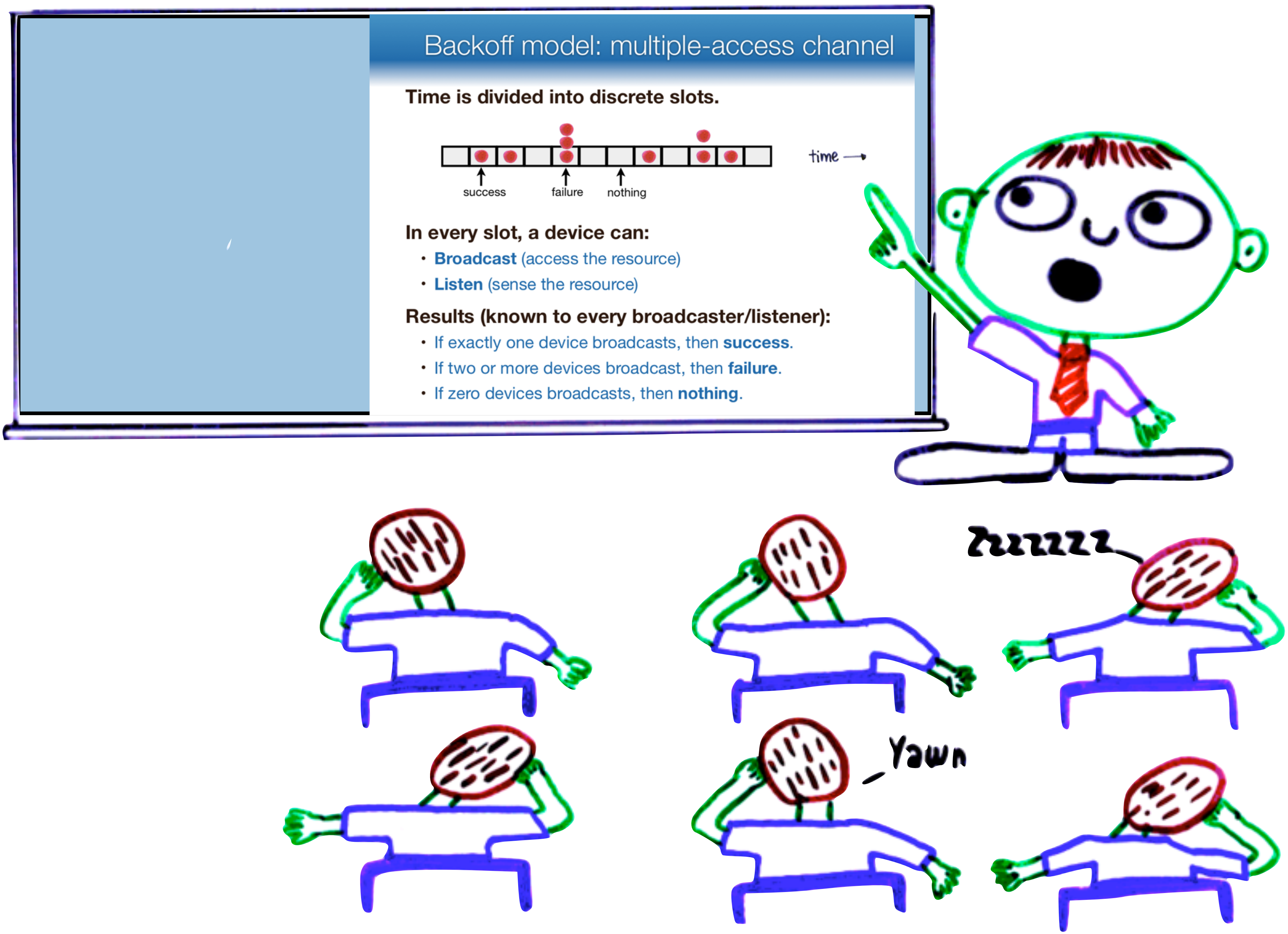
In every slot, a device can:

- **Broadcast** (access the channel)
- **Listen** (sense the channel)

Results (known to every broadcaster/listener):

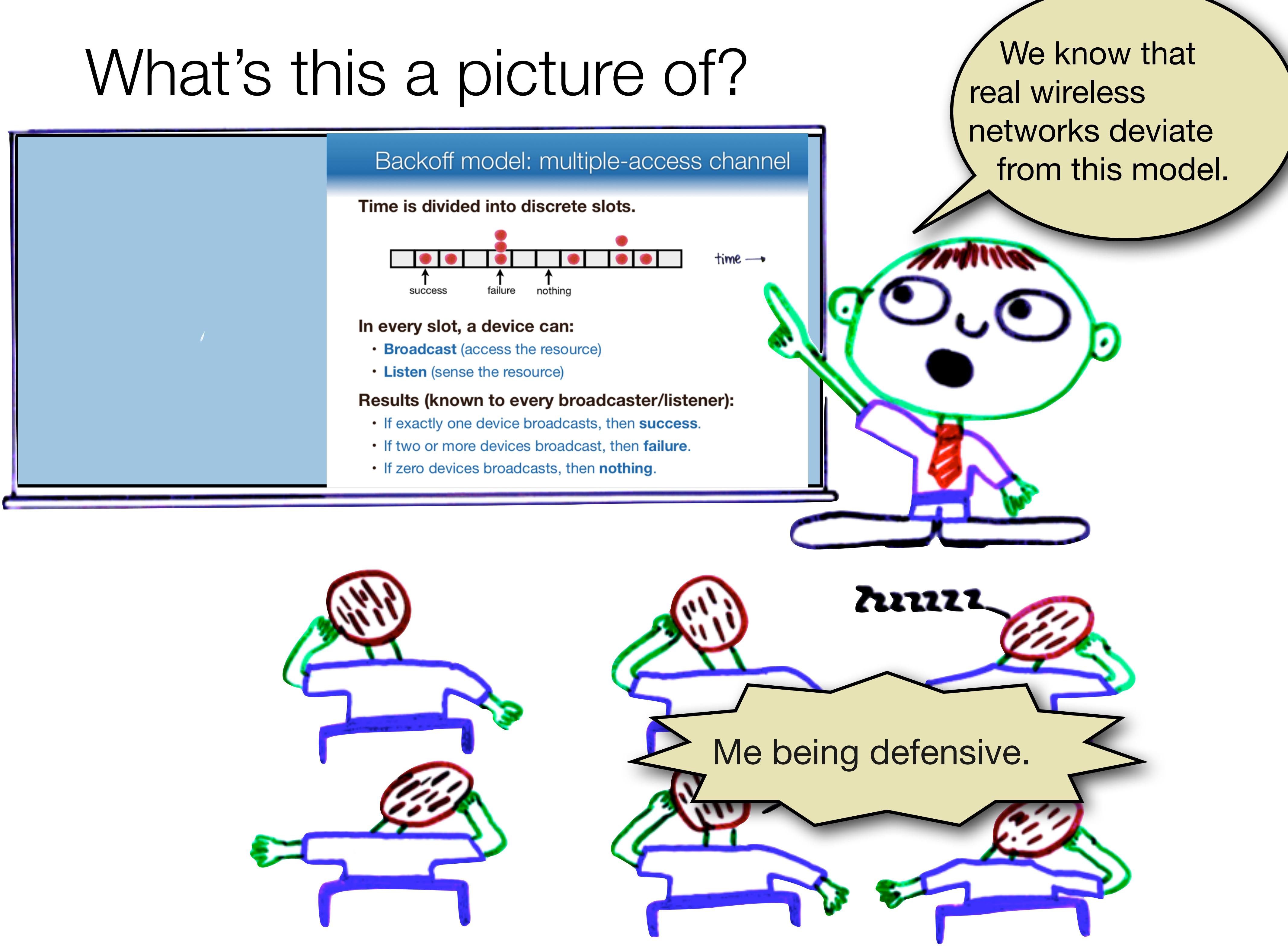
- If **exactly one** device broadcasts, then **success**.
- If **two or more** devices broadcast, then **failure**.
- If **zero** devices broadcast, then **nothing**.

What's this a picture of?

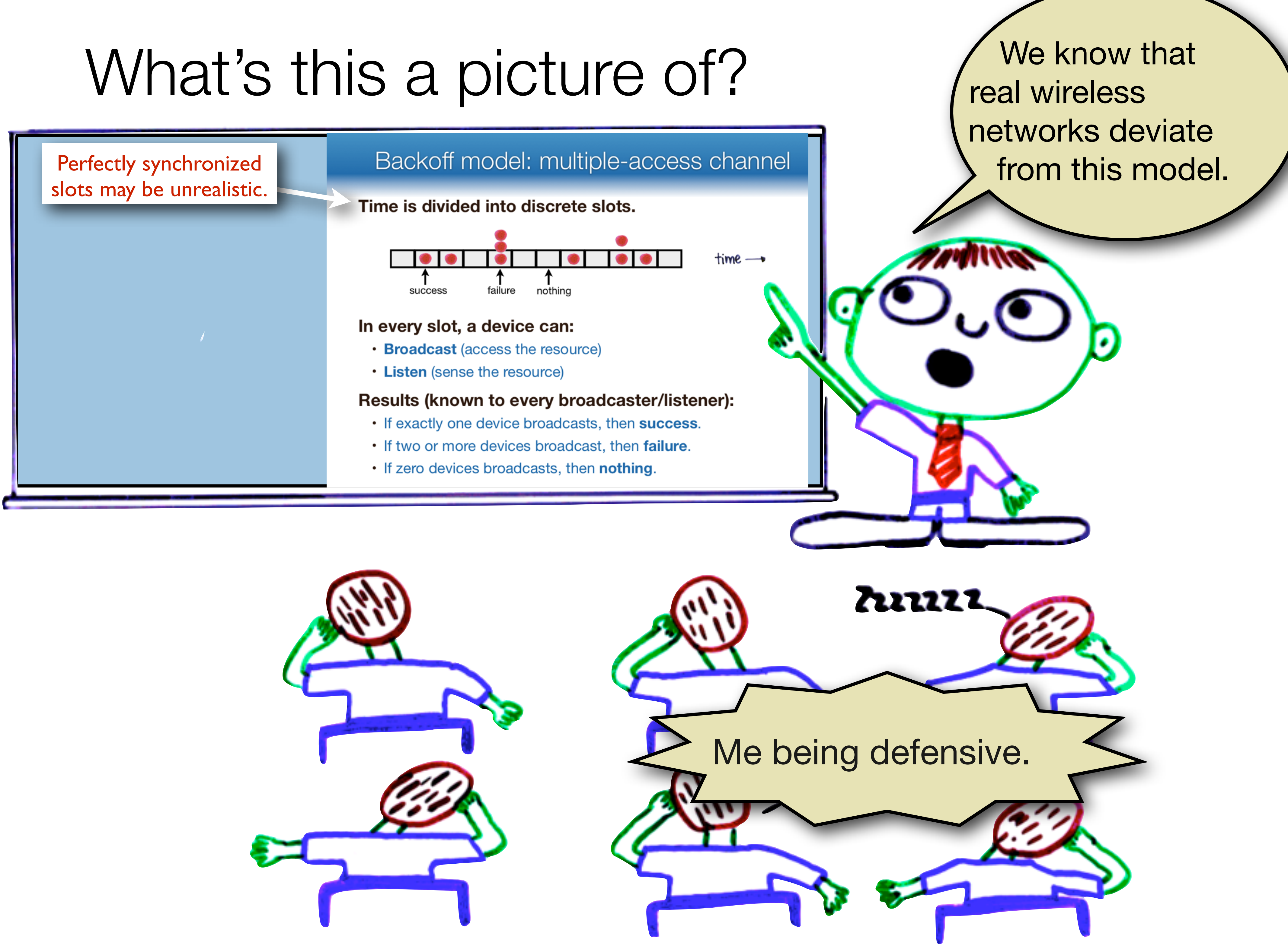


What's this a picture of?

What's this a picture of?



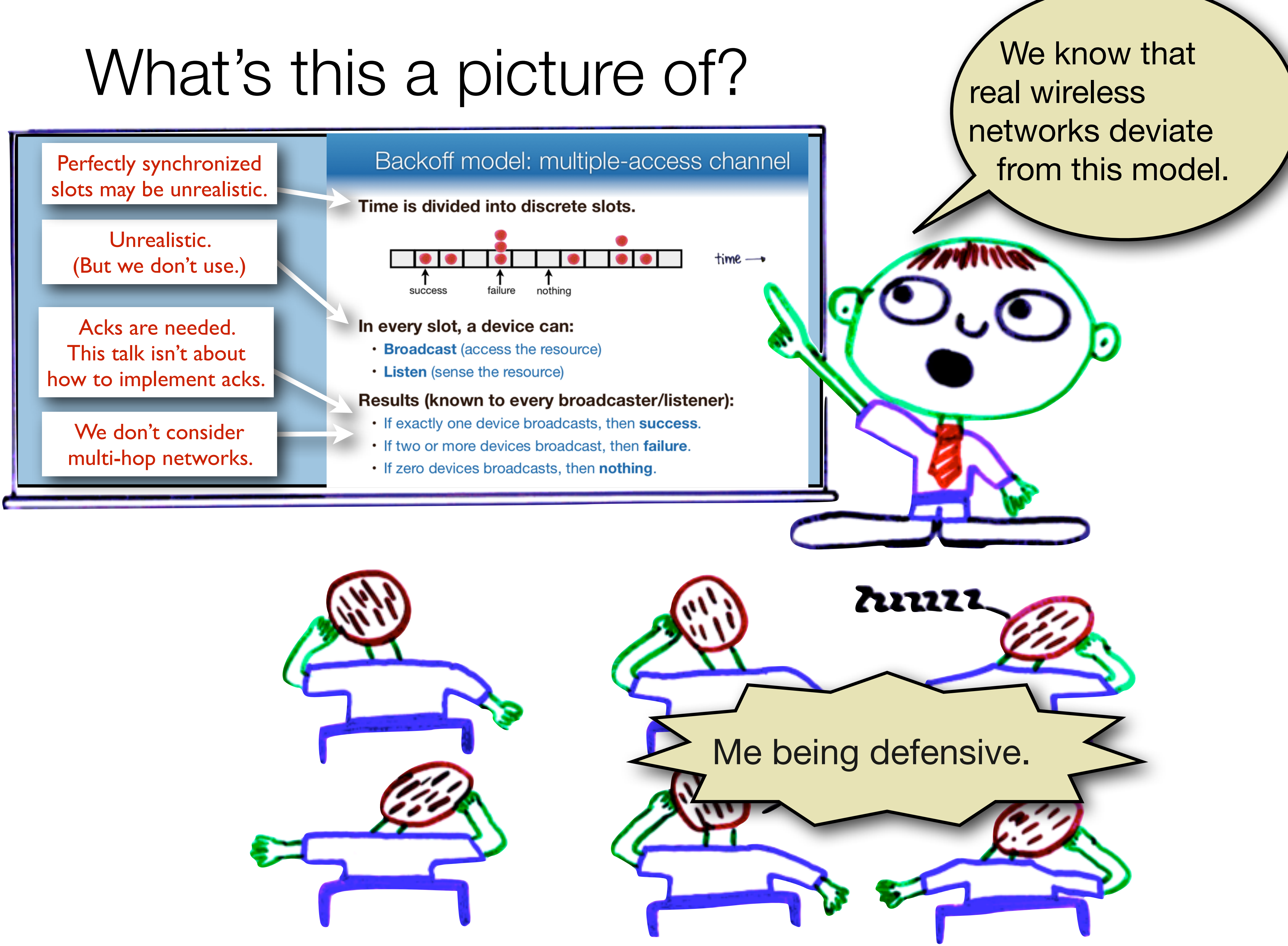
What's this a picture of?



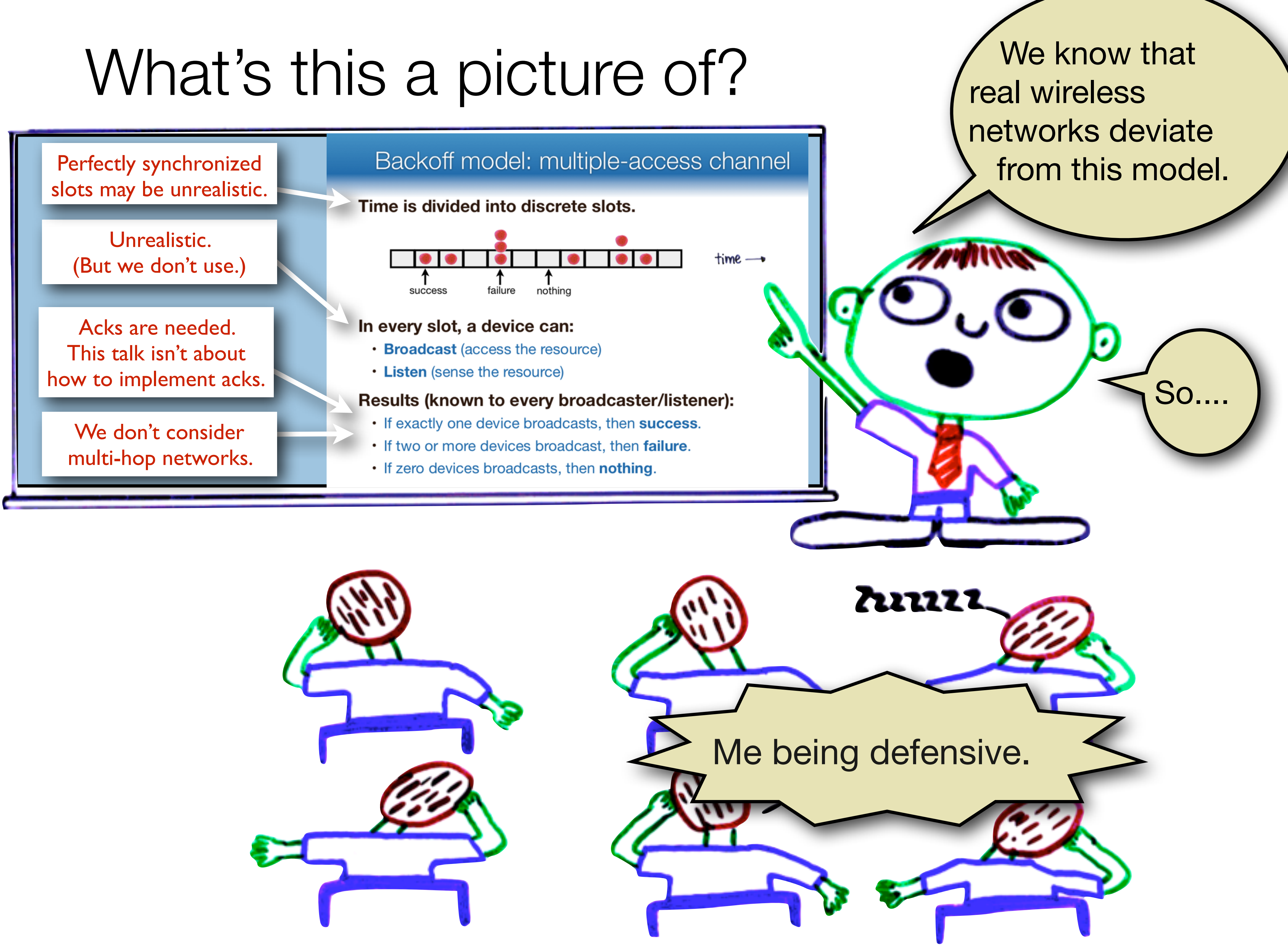
What’s this a picture of?

What’s this a picture of?

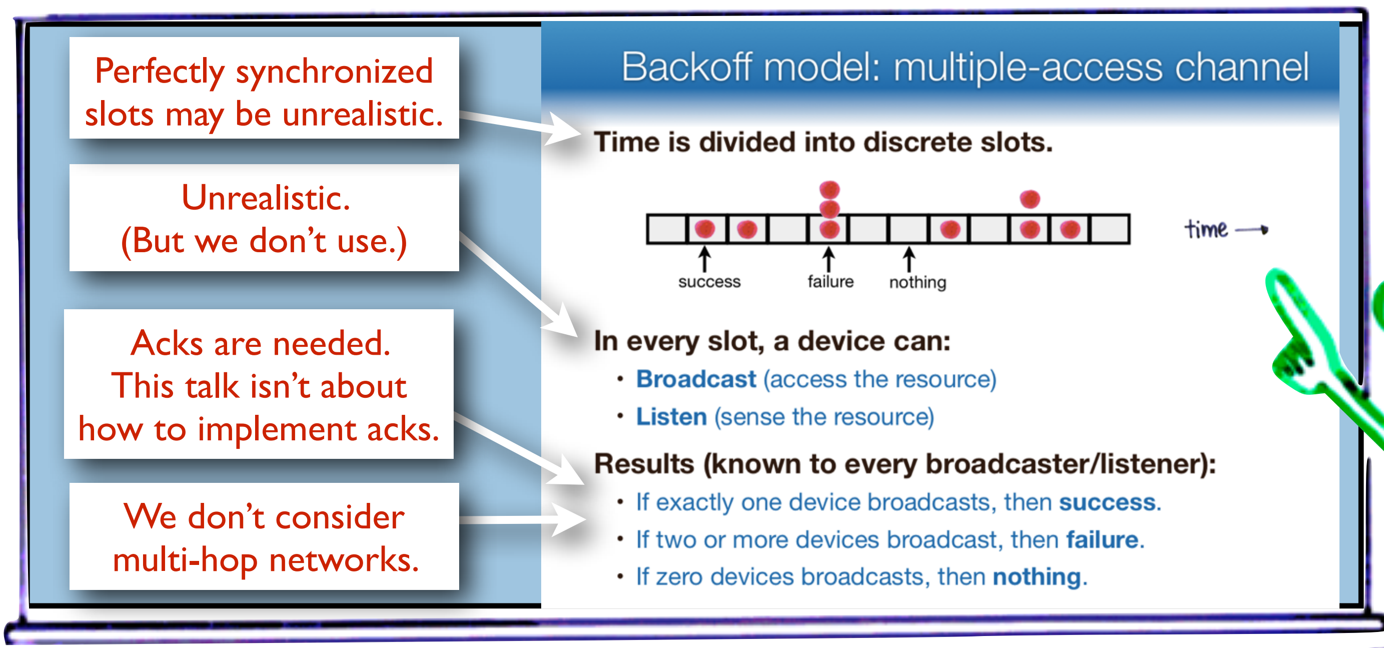
What’s this a picture of?

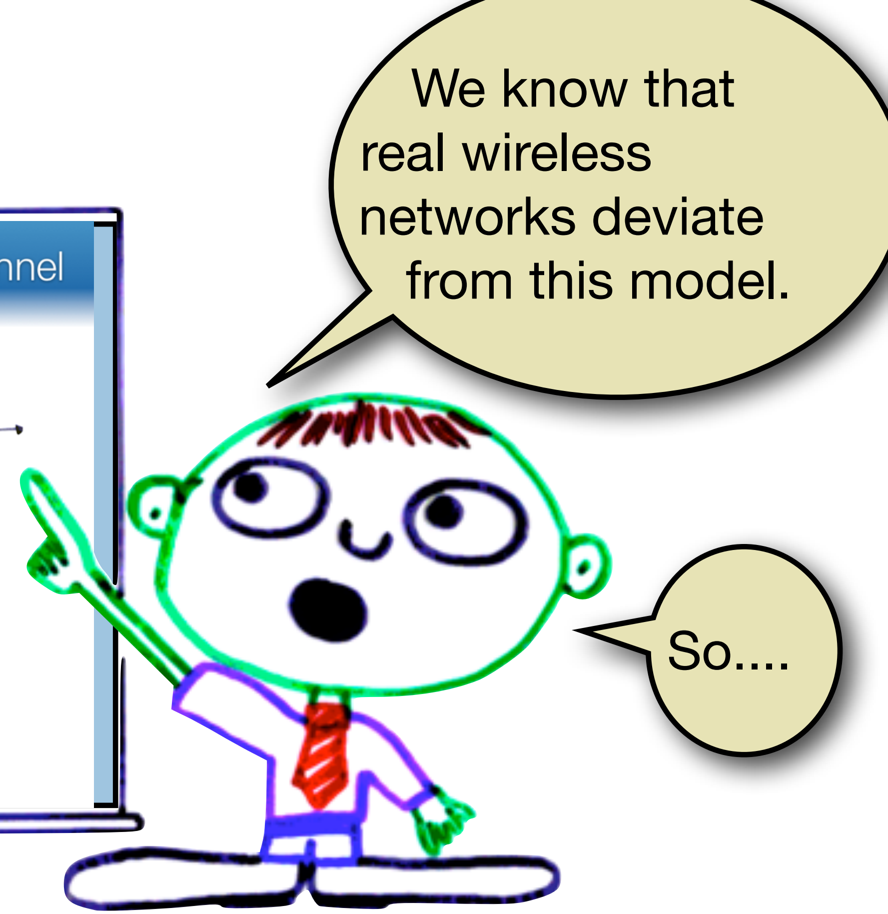


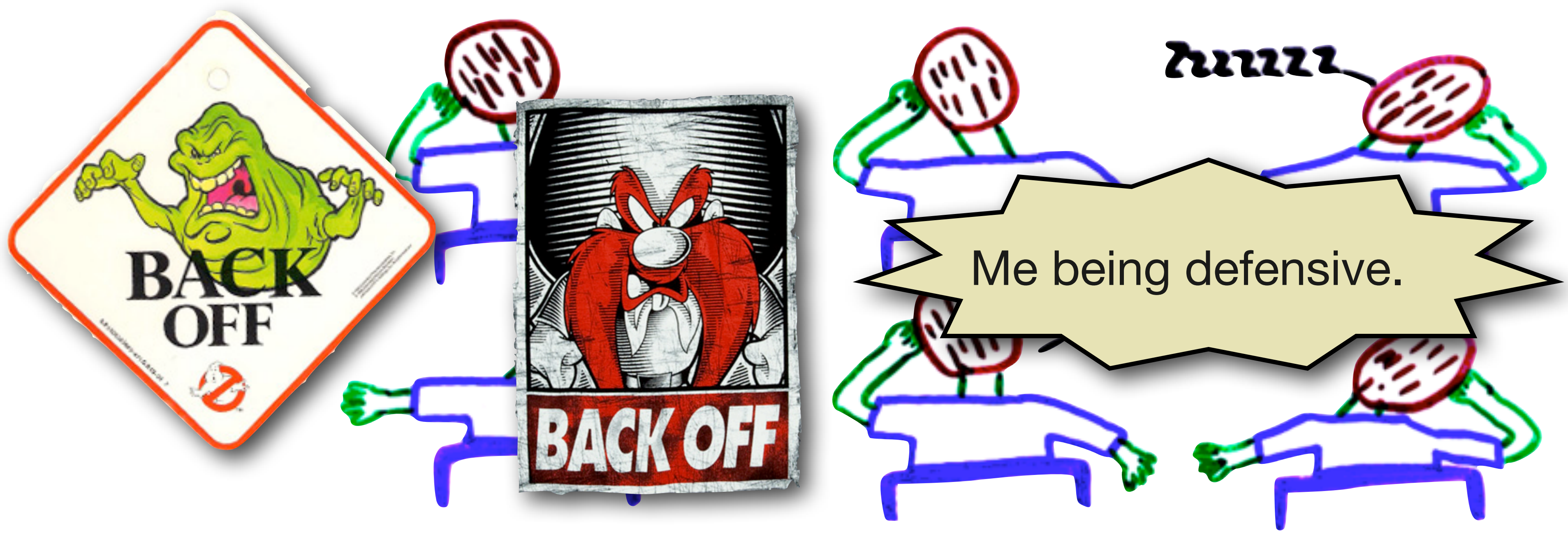
What’s this a picture of?



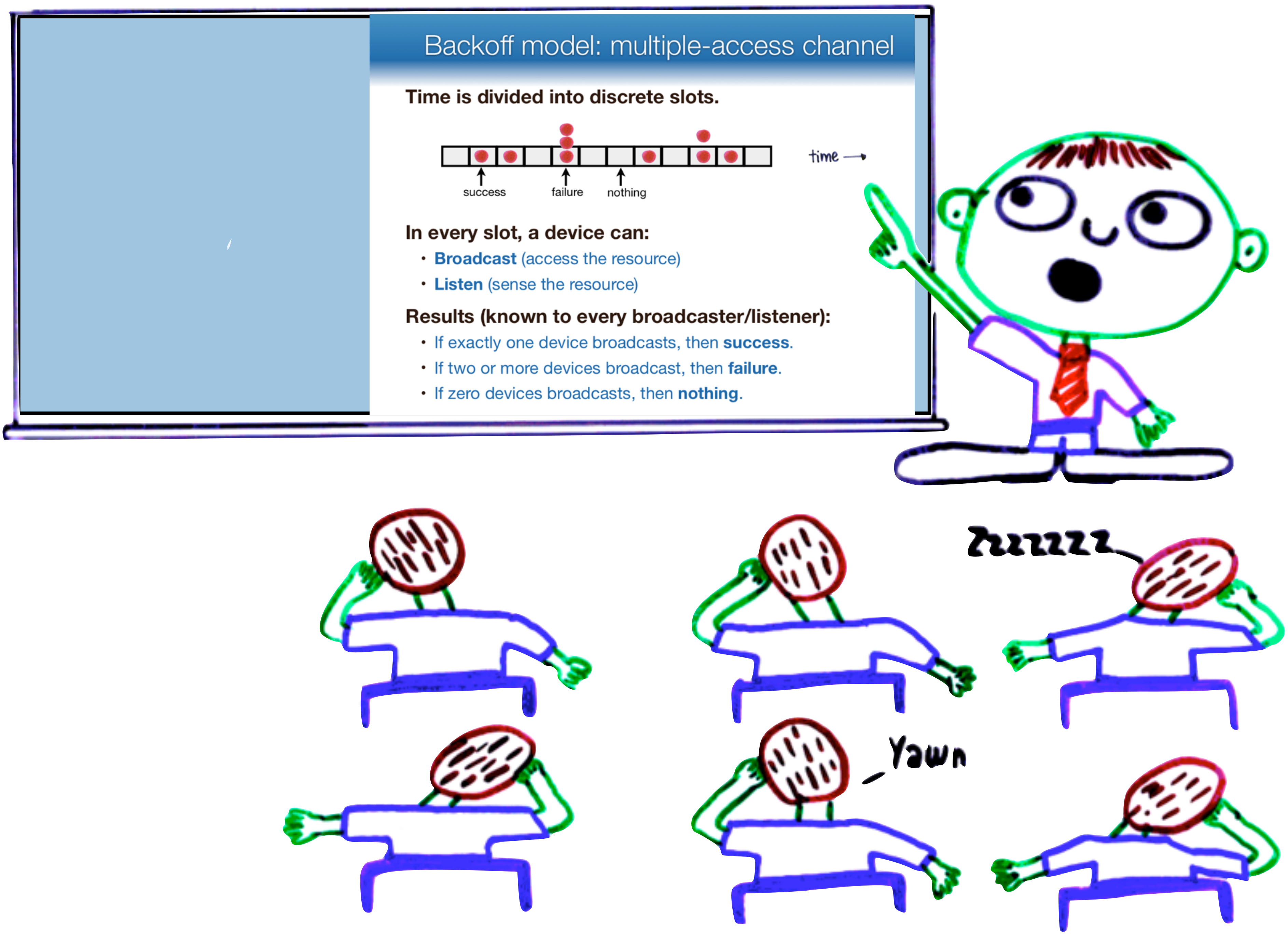
What's this a picture of?







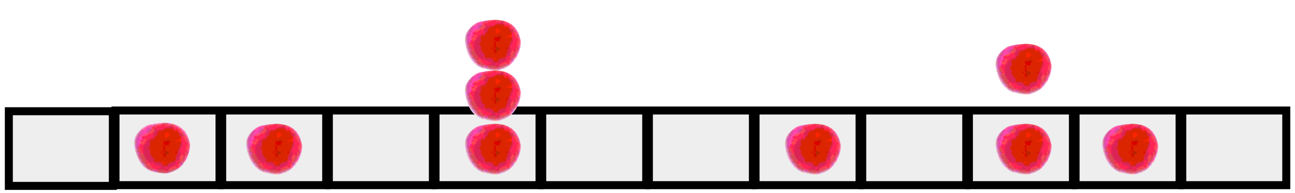
I want to focus on backoff as a theory problem.



1. Throughput

(first backoff dilemma)

$$\frac{\#\ \text{successful slots}}{\text{total number of slots}}$$



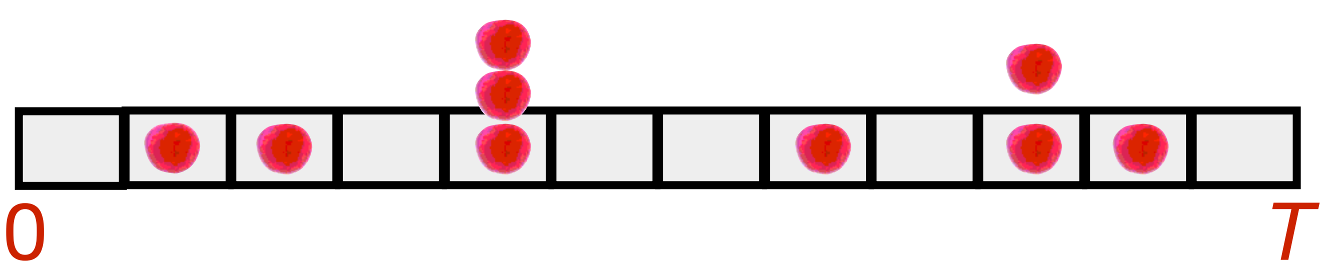
throughput = 4/12

Next few slides: batch scenario

All n packets start at the same time $t = 0$.

Let T = running time
(= time when last request succeeds).

Throughput: n/T.



throughput = 4/12

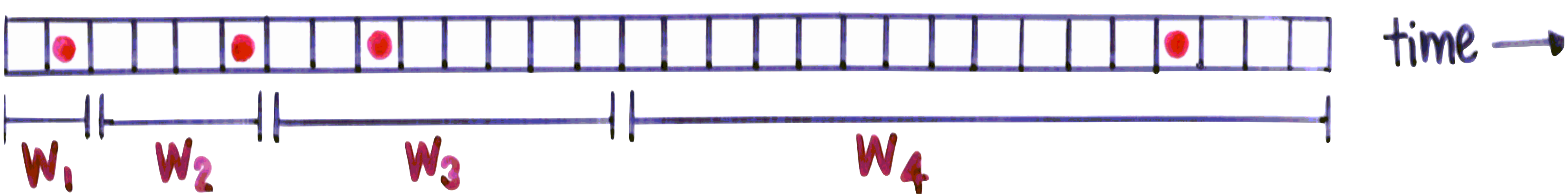
Standard binary exponential backoff

Window size *W* = 2

Repeat until successful transmission:

- Randomly choose slot *t* in window.
- Try to broadcast at slot *t*.
- If collision, wait to end of *W*.
  Then double *W*.

Why double?
What if the window size
changes by a different factor?



What backoff rate is best for batches?

Constant-sized windows

- *W* is a fixed constant

back off slowly

Binary exponential growth

- After collision: $W = 2W$

back off rapidly

What backoff rate is best for batches?

Constant-sized windows

- *W* is a fixed constant

Additive increase

- After collision: $W = W + 1$

back off slowly

Binary exponential growth

- After collision: $W = 2W$

back off rapidly

What backoff rate is best for batches?

Constant-sized windows

- *W* is a fixed constant

Additive increase

- After collision: $W = W + 1$

Logarithmic growth

- After collision: $W = W\left(1 + \frac{1}{\log W}\right)$

Binary exponential growth

- After collision: $W = 2W$

back off slowly

back off rapidly

What backoff rate is best for batches?

Constant-sized windows

- *W* is a fixed constant

Additive increase

- After collision: $W = W + 1$

Logarithmic growth

- After collision: $W = W\left(1 + \frac{1}{\log W}\right)$

LogLog growth

- After collision: $W = W\left(1 + \frac{1}{\log \log W}\right)$

Binary exponential growth

- After collision: $W = 2W$

back off slowly

back off rapidly

What backoff rate is best for batches?

Approx. running time

Constant-sized windows

- *W* is a fixed constant

exponential in n

Additive increase

- After collision: $W = W + 1$

$\tilde{O}(n^2)$

Logarithmic growth

- After collision: $W = W\left(1 + \frac{1}{\log W}\right)$

$\tilde{O}(n \log n)$

LogLog growth

- After collision: $W = W\left(1 + \frac{1}{\log \log W}\right)$

$\tilde{O}(n \operatorname{loglog} n)$

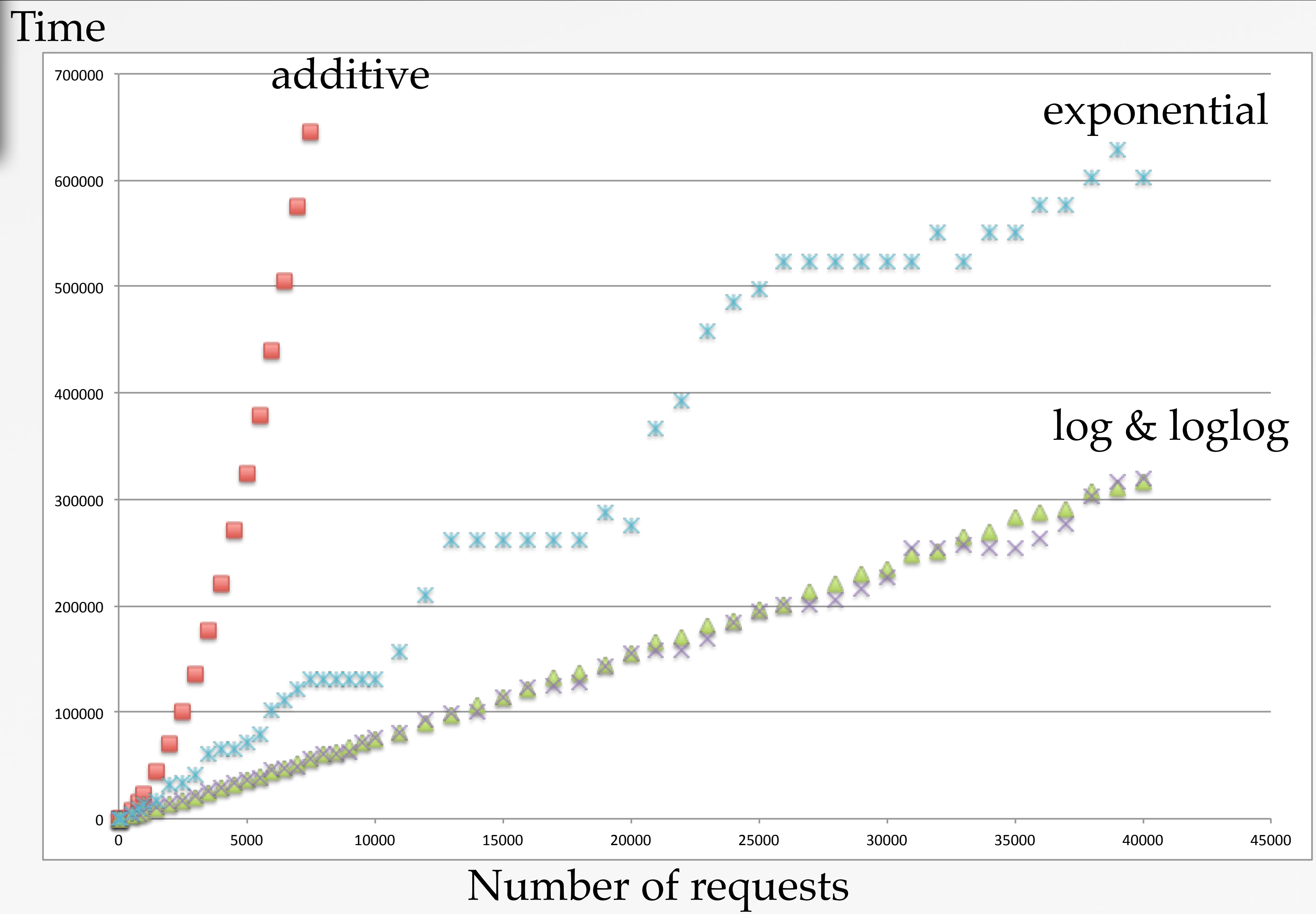
Binary exponential growth

- After collision: $W = 2W$

$\tilde{O}(n \log n)$

[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

Comparison [Gilbert 14]



What backoff rate is best for batches?

Actual running time

Constant-sized windows

- W is a fixed constant

exponential in n

Additive increase

- After collision: $W = W + 1$

$O(n^2/\log n)$

Logarithmic growth

- After collision: $W = W\left(1 + \frac{1}{\log W}\right)$

$O(n \log n / \log\log n)$

LogLog growth

- After collision: $W = W\left(1 + \frac{1}{\log\log W}\right)$

$O(n \log\log n / \log\log\log n)$

Binary exponential growth

- After collision: $W = 2W$

$O(n \log n)$

[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

What backoff rate is best for batches?

Actual running time

Optimal (monotonic):
O(n loglog n / logloglog n)

Cons

- W

exponential in n

Addit

- After collision: $W = W + 1$

O(n^2/log n)

Logarithmic growth

- After collision: $W = W\left(1 + \frac{1}{\log W}\right)$

O(n log n / loglog n)

LogLog growth

- After collision: $W = W\left(1 + \frac{1}{\log\log W}\right)$

O(n loglog n / logloglog n)

Binary exponential growth

- After collision: $W = 2W$

O(n log n)

[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

Moral of the story

Exponential backoff is disappointing

- Used everywhere.
- Poor throughput: < 1/polylog(n).
- Example experiment: n=100.
  - About 10% of slots are used.
  - About 90% of resource is wasted!

LogLog backoff is better

- In simple experiments, much better.
- It's the best monotonic backoff for batch arrivals.
  [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]
- Still, it ***cannot*** achieve ***constant*** throughput.

Moral of the story

Exponential backoff is disappointing

- Used everywhere.
- Poor throughput: $< 1/\mathrm{polylog}(n)$.
- Example experiment: n=100.
  - About 10% of slots are used.
  - About 90% of resource is wasted!

LogLog backoff is better

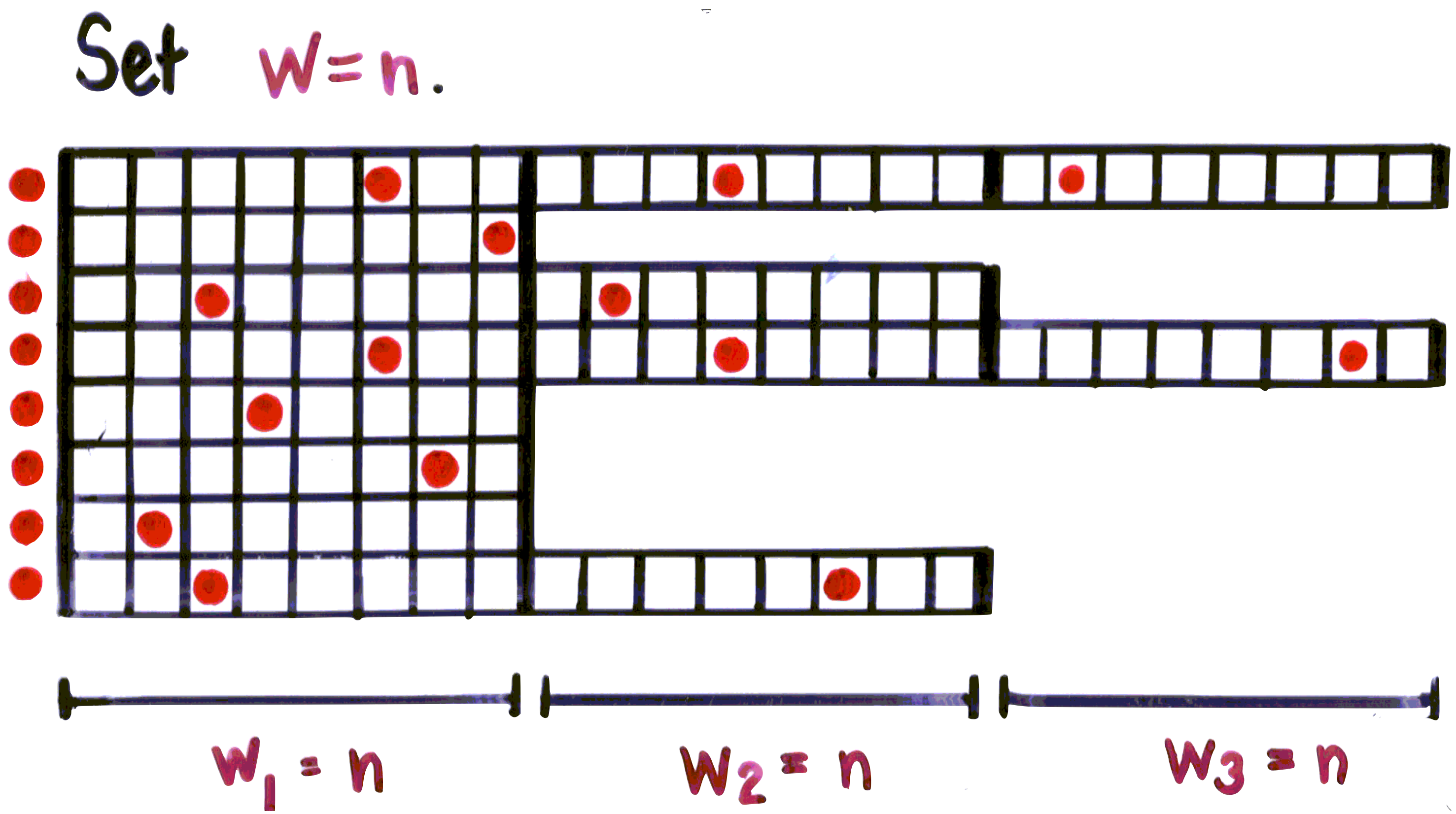
- In simple experiments, much better.
- It's the best monotonic backoff for batch arrivals.
  [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]
- Still, it ***cannot*** achieve ***constant*** throughput.

Next: explanation why....

Simple batch example: we know n (# packets)

Goal: explain why exp backoff backs off too quickly on batches.

fixed-sized windows

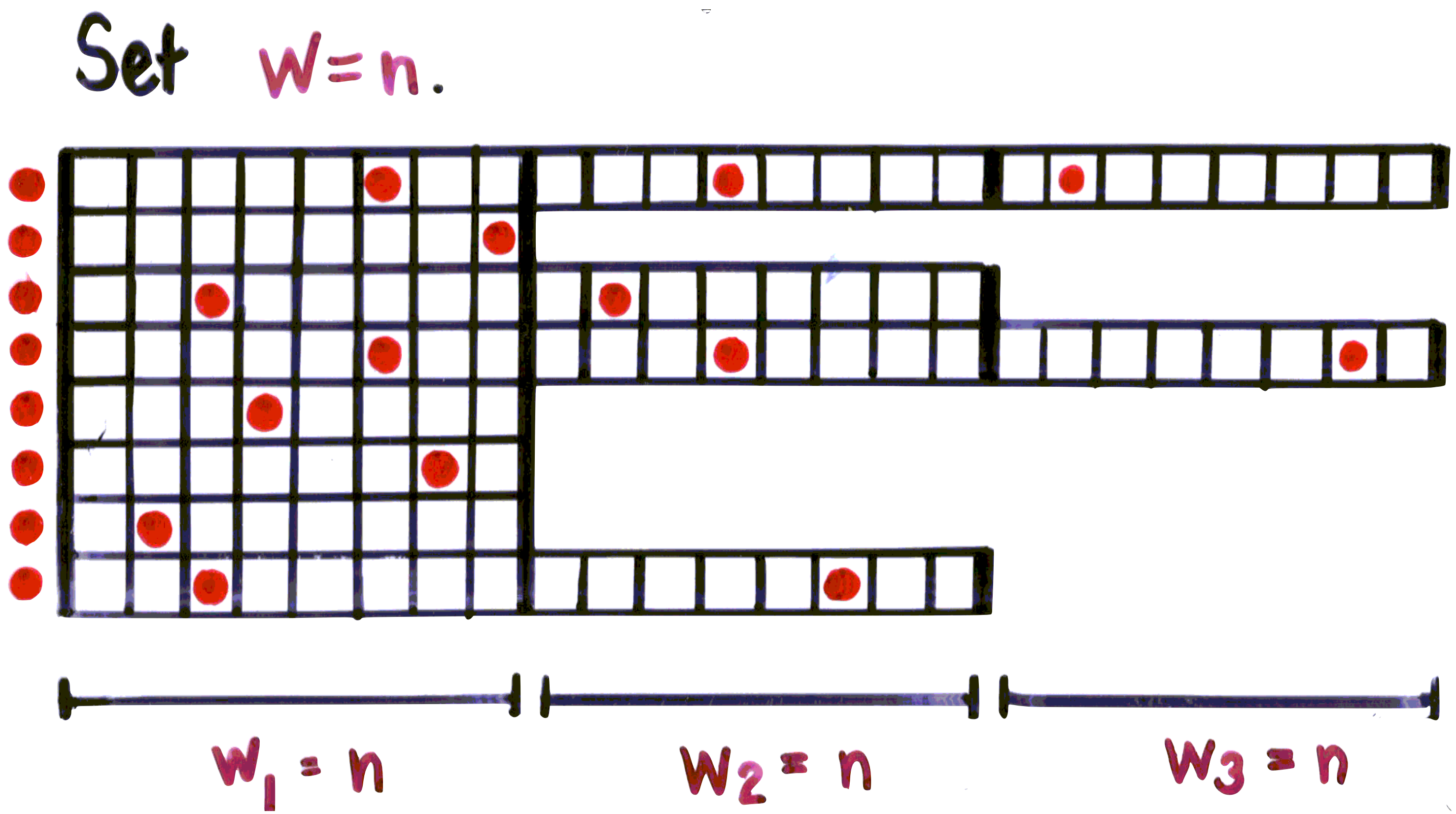


Claim: W.h.p, all packets transmit in lglg $n \pm O(1)$ rounds.

Simple batch example: we know *n* (# packets)

Goal: explain why exp backoff backs off too quickly on batches.

fixed-sized windows

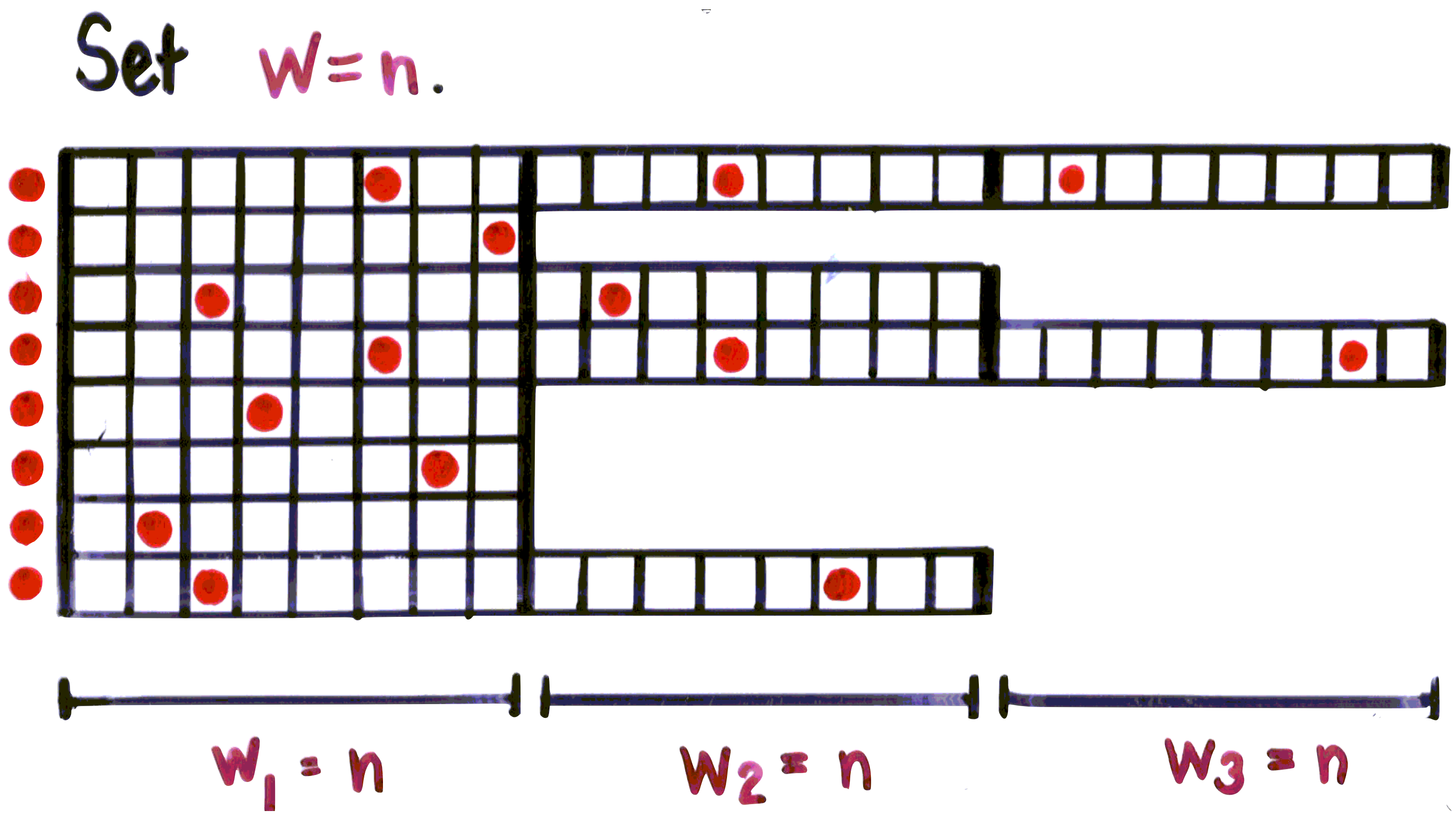


Claim: W.h.p, all packets transmit in lglg $n \pm O(1)$ rounds.

running time = n lglg $n + O(n)$

Simple batch example: we know *n* (# packets)

Goal: explain why exp backoff backs off too quickly on batches.



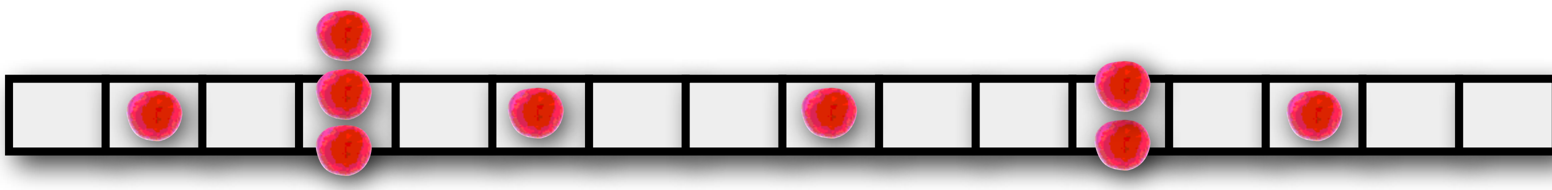
Claim: W.h.p, all packets transmit in lglg $n \pm O(1)$ rounds.

running time = n lglg $n + O(n)$ $\Rightarrow$ throughput = $O(1/$ lglg $n)$

Intuition for Fixed Backoff (size-n windows)

Collision probs square (decrease) in each round.

(1) $n/2$ packets ⇒ Pr[collision]≤1/2.

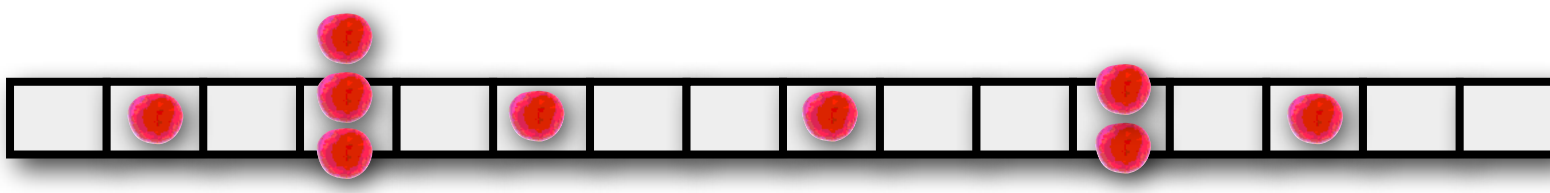


(2) E[#packets remaining] ≤$n/4$

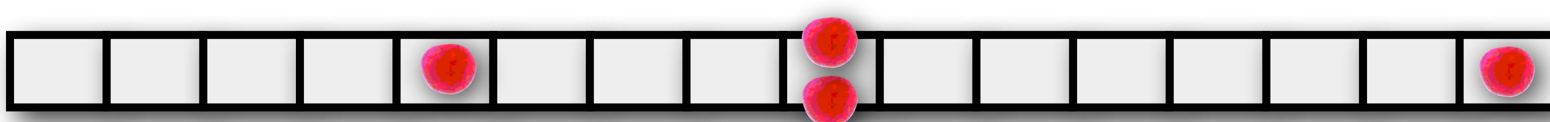
Intuition for Fixed Backoff (size-n windows)

Collision probs square (decrease) in each round.

(1) $n/2$ packets $\Rightarrow$ Pr[collision]$\leq 1/2$.



(2) E[#packets remaining] $\leq n/4$ $\Rightarrow$ Pr[collision]$\leq 1/4$.

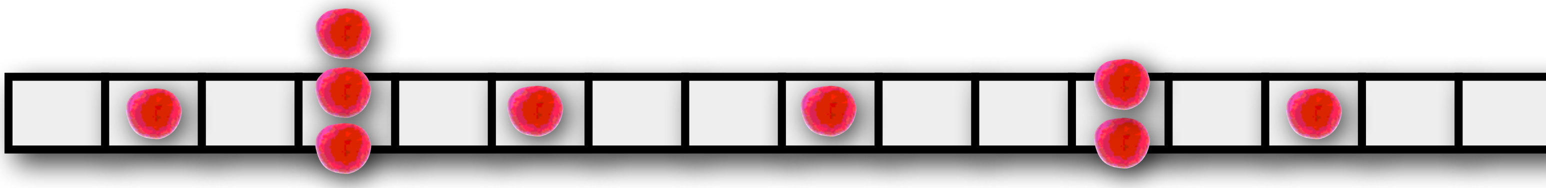


(3) E[#packets remaining]$\leq n/16$

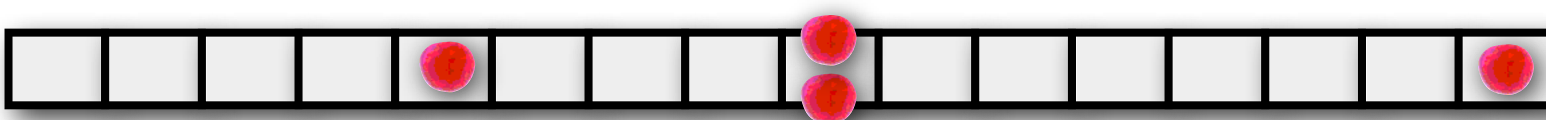
Intuition for Fixed Backoff (size-n windows)

Collision probs square (decrease) in each round.

(1) $n/2$ packets $\Rightarrow$ Pr[collision]$\leq 1/2$.



(2) E[#packets remaining] $\leq n/4$ $\Rightarrow$ Pr[collision]$\leq 1/4$.



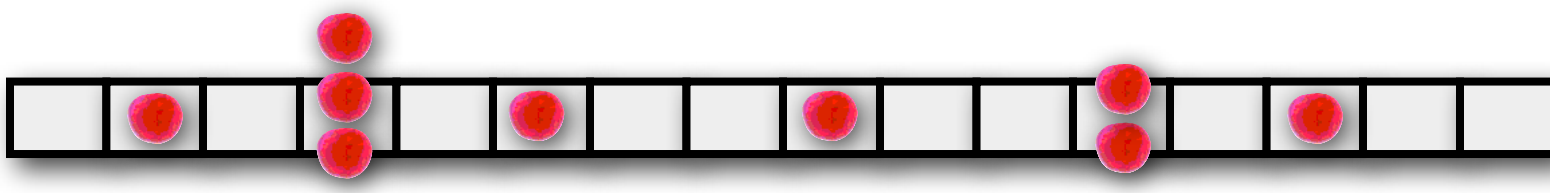
(3) E[#packets remaining]$\leq n/16$ $\Rightarrow$ Pr[collision]$\leq 1/16$.

(4) E[#packets remaining]$\leq n/256$

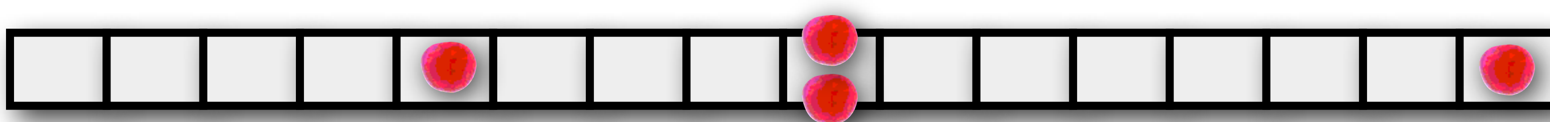
Intuition for Fixed Backoff (size-n windows)

Collision probs square (decrease) in each round.

(1) $n/2$ packets $\Rightarrow$ Pr[collision]$\leq 1/2$.



(2) E[#packets remaining] $\leq n/4$ $\Rightarrow$ Pr[collision]$\leq 1/4$.



(3) E[#packets remaining]$\leq n/16$ $\Rightarrow$ Pr[collision]$\leq 1/16$.

(4) E[#packets remaining]$\leq n/256$

Good intuition, but incorrect argument!

Analysis of Exponential Backoff

[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

Almost no packets transmit until $W_k = \Theta(n)$.

Exponential backoff still uses $\lg\lg n \pm \Theta(1)$ rounds.

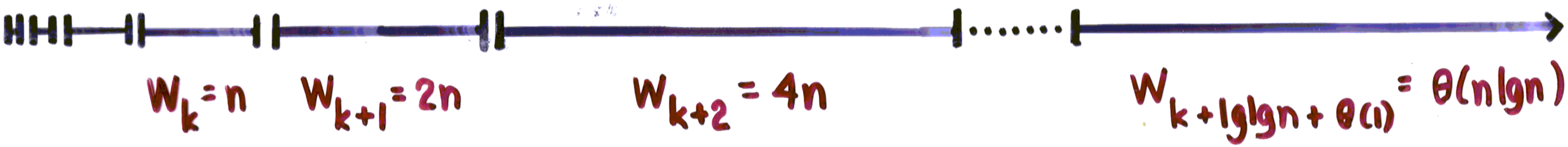
$\Rightarrow \Theta(n \lg n)$ time.

$\Rightarrow$ exponential backoff backs off too quickly on bursts.

Analysis of Exponential Backoff

[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

Exponential backoff is exquisitely sensitive to constants.



2-exponential backoff ($W_{k+1} = 2W_k$) has $\Theta(n \lg n)$ time.

4-exponential backoff ($W_{k+1} = 4W_k$) has $\Theta(n \lg^2 n)$ time.

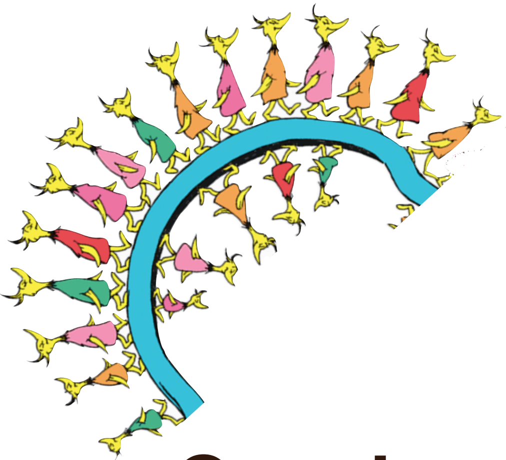
r-exponential backoff ($W_{k+1} = rW_k$) has $\Theta(n \lg^{\lg r} n)$ time.

Summary for batch arrivals

Exponential backoff backs off too quickly on batches.

Backing off more slowly is opt for monotonic backoff.

It is possible to get asymptotically optimal backoff if we sometimes back off and sometimes back on.



Next few slides: dynamic arrivals

(packets start at arbitrary times)

Queuing theory (with Poisson arrivals)

[Hastad, Leighton, Rogoff 87] [Goodman, Greenberg, Madras 88] [Goldberg and MacKenzie 96] [Raghavan and Upfal 99][Goldberg, Mackenzie, Paterson, Srinivasan 00]

- Goal: achieve *stability* with good arrival rates.
- *Exponential backoff* is not as stable as *polynomial backoff*.

Adversarial queuing theory arrivals

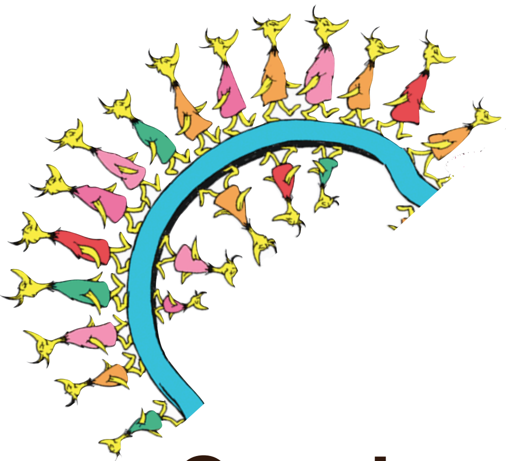
[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

- Exponential backoff does not adapt well to bursts.

Adversarial queueing theory with *n* fixed stations

[Chlebus, Kowalski, Rokicki 06 12] [Anantharamu, Chlebus, Rokicki 09] [Chlebus, Kowalski 04] [Chlebus, Gasieniec, Kowalsi, Radzik 05] [Chrobak, Gasieniec, Kowalski 07] etc

- Adversarial injections
- Often deterministic algorithms: round-robin/binary search/etc.



Next few slides: dynamic arrivals

(packets start at arbitrary times)

Queuing theory (with Poisson arrivals)

[Hastad, Leighton, Rogoff 87] [Goodman, Greenberg, Madras 88] [Goldberg and MacKenzie 96] [Raghavan and Upfal 99][Goldberg, Mackenzie, Paterson, Srinivasan 00]

- Goal: achieve *stability* with good arrival rates.
- *Exponential backoff* is not as stable as *polynomial backoff*.

Adversarial queuing theory arrivals

packet-centric versus station-centric view of backoff.

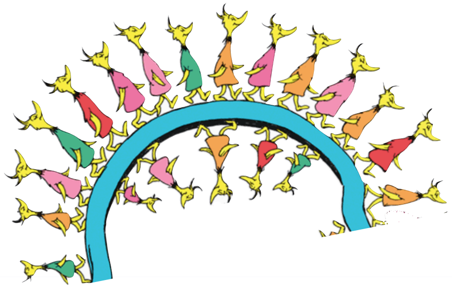
[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

- Exponential backoff does not adapt well to bursts.

Adversarial queueing theory with *n* fixed stations

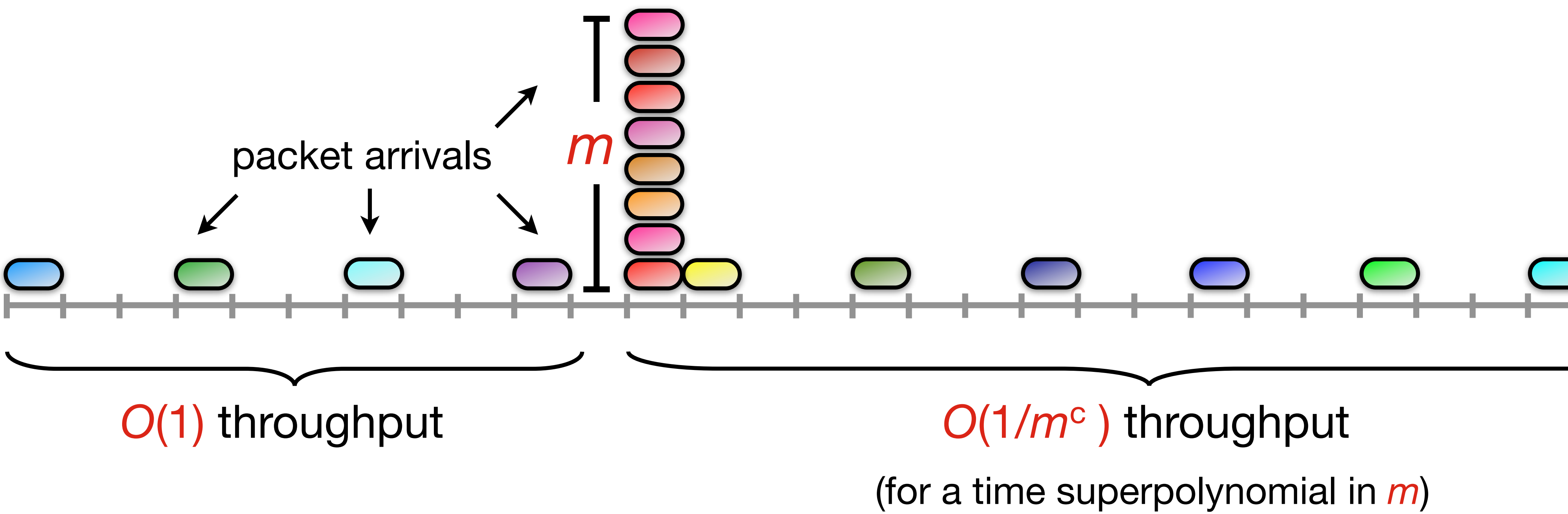
[Chlebus, Kowalski, Rokicki 06 12] [Anantharamu, Chlebus, Rokicki 09] [Chlebus, Kowalski 04] [Chlebus, Gasieniec, Kowalsi, Radzik 05] [Chrobak, Gasieniec, Kowalski 07] etc

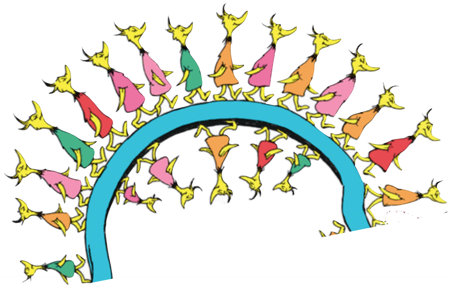
- Adversarial injections
- Often deterministic algorithms: round-robin/binary search/etc.



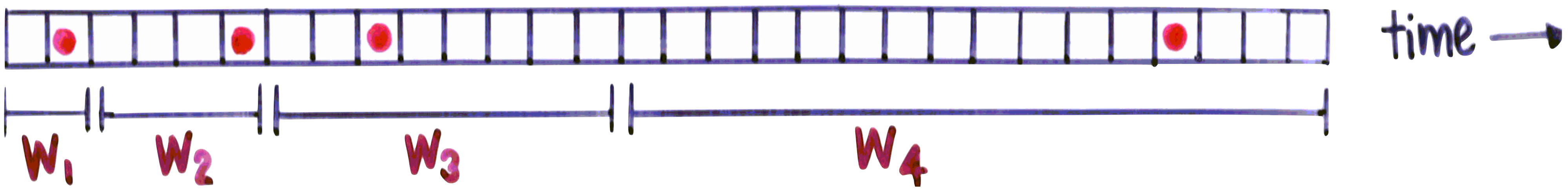
Exponential backoff and bursts

Exponential backoff may not recover from bursts for a time superpolynomial in the size of the burst. [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]





Exponential backoff and bursts

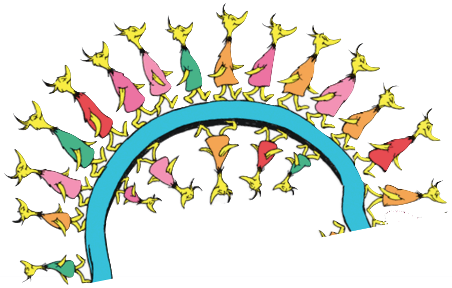


Broadcast probability

- A packet in the system for ***d*** time units broadcasts with probability $\Theta(1/d)$.

Contention at time *t*

- The contention at time ***t*** is the sum of the broadcast probabilities of all packets currently in the system.



Exponential backoff and bursts

Contention at time t

- The contention at time t is the sum of the access probabilities of all jobs currently in the system.

contention c = O(1)

- prob(slot t is successful) = O(1)

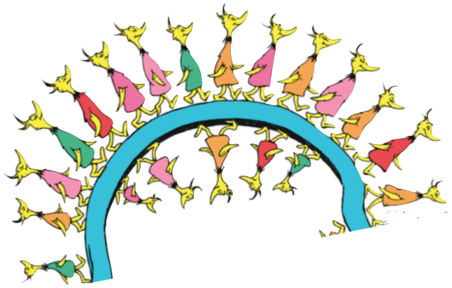
contention c = Ω(1)

- prob(the slot is successful) = $2^{-\Theta(c)}$

The success probability is exponentially small in the contention.

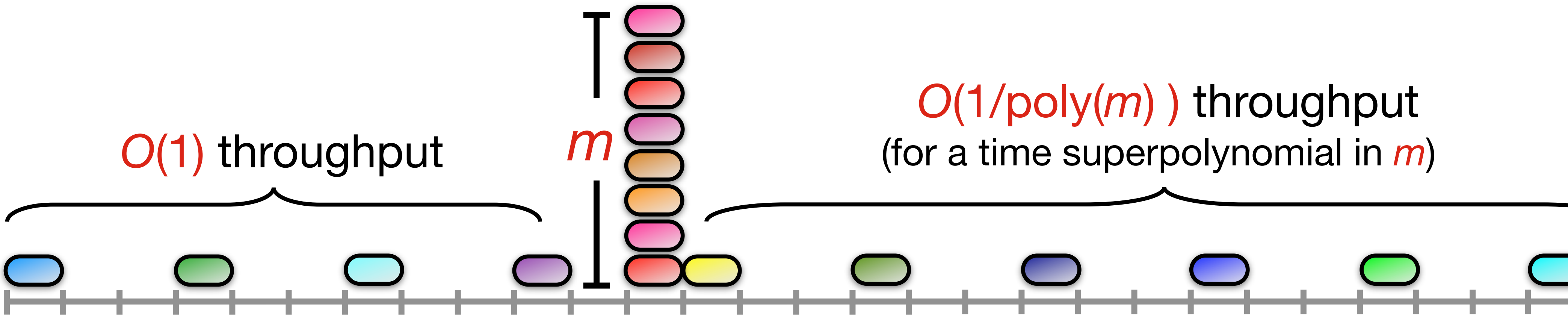
contention c = o(1)

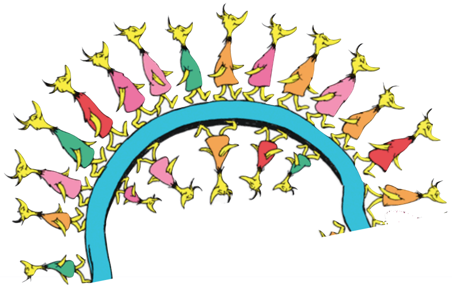
- prob(slot is not empty) = Θ(c)



Exponential backoff and bursts

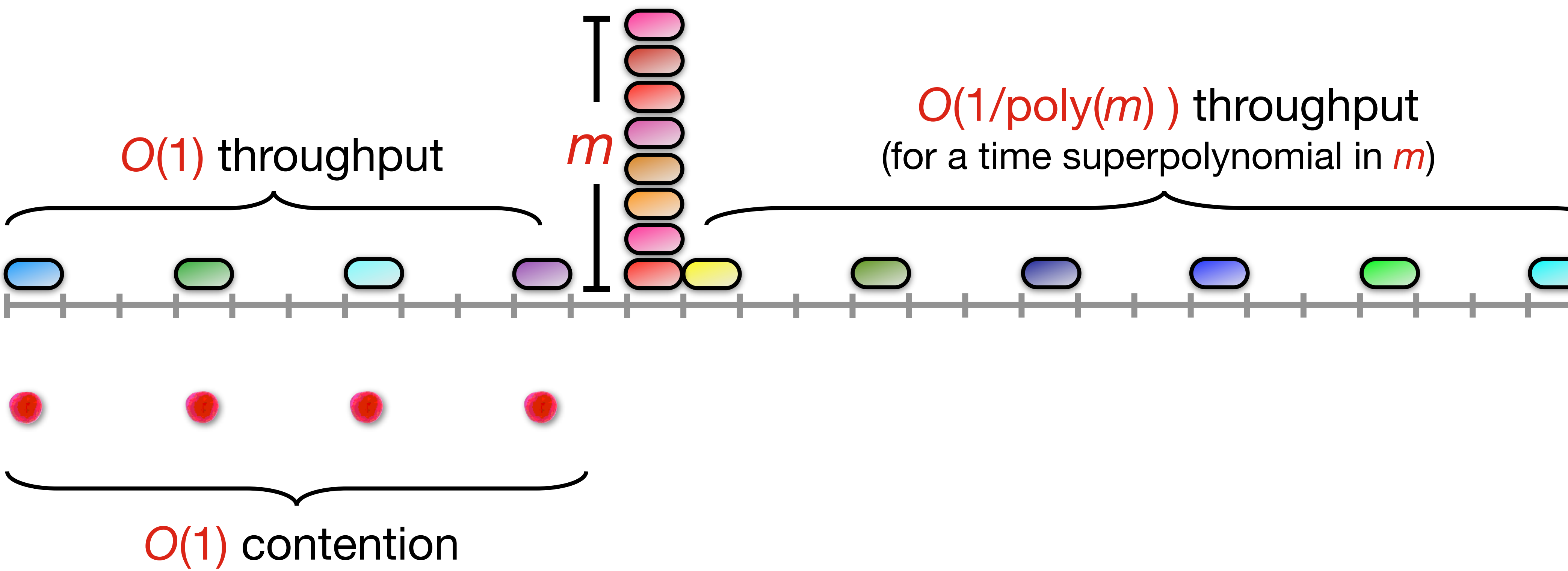
Exponential backoff may not recover from bursts for a time superpolynomial in the size of the burst. [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

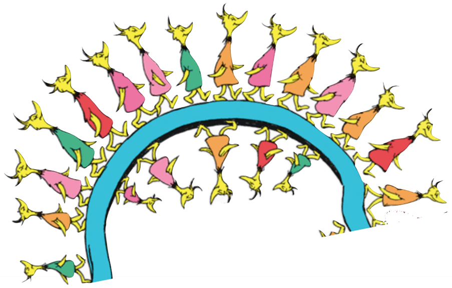




Exponential backoff and bursts

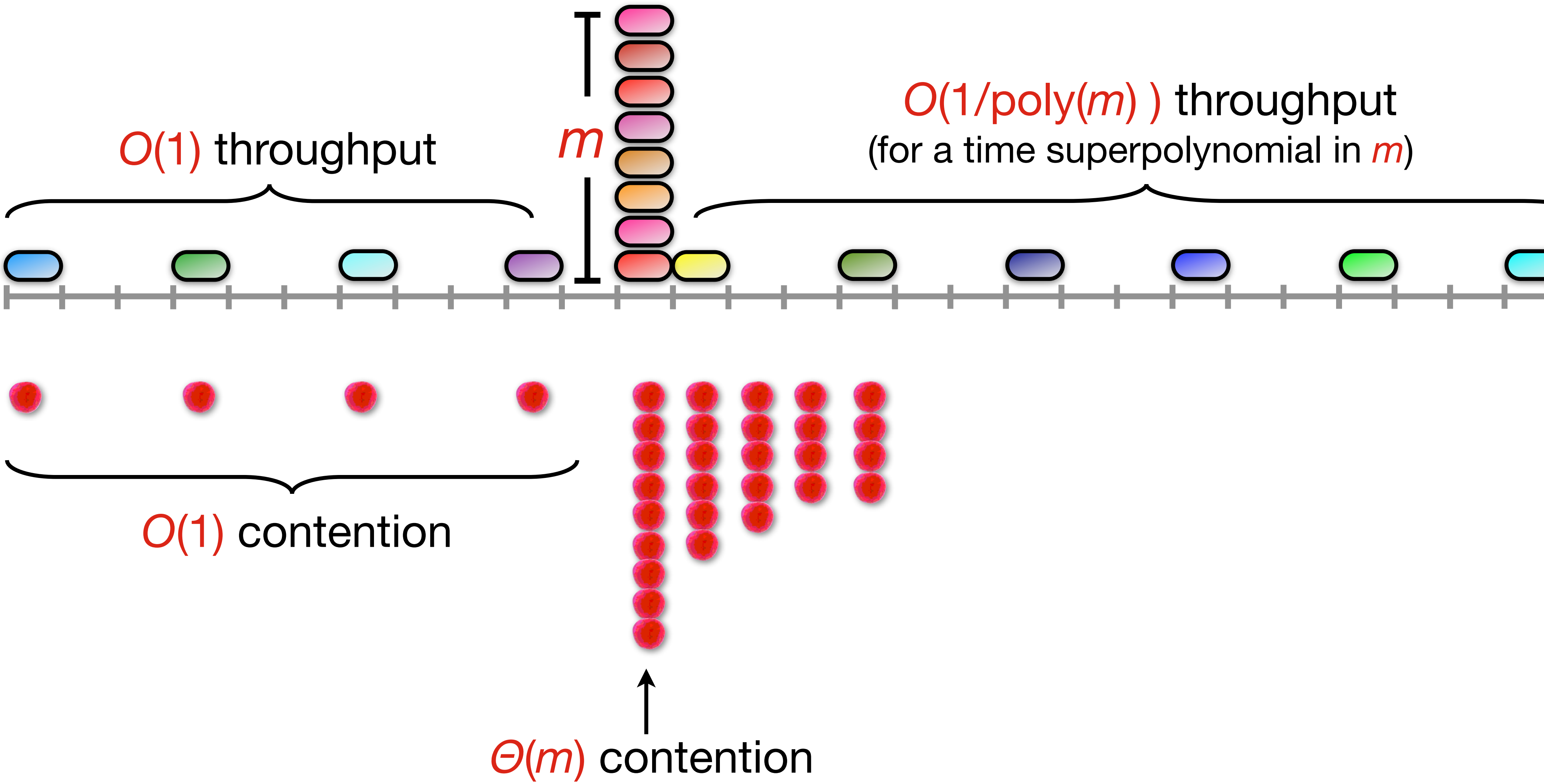
Exponential backoff may not recover from bursts for a time superpolynomial in the size of the burst. [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

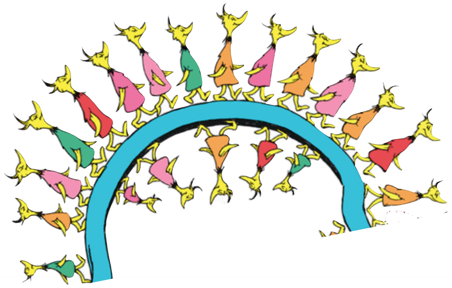




Exponential backoff and bursts

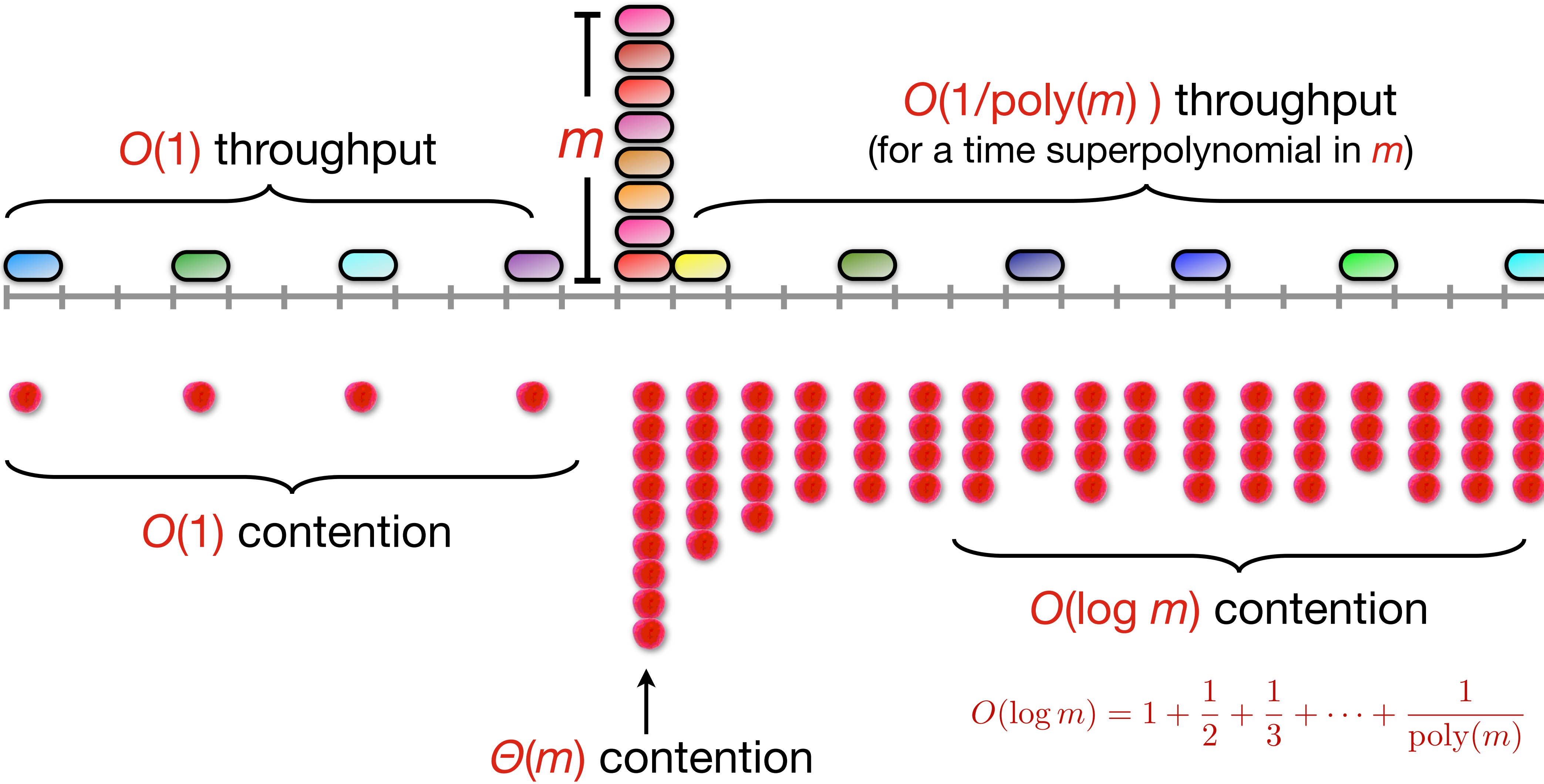
Exponential backoff may not recover from bursts for a time superpolynomial in the size of the burst. [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]





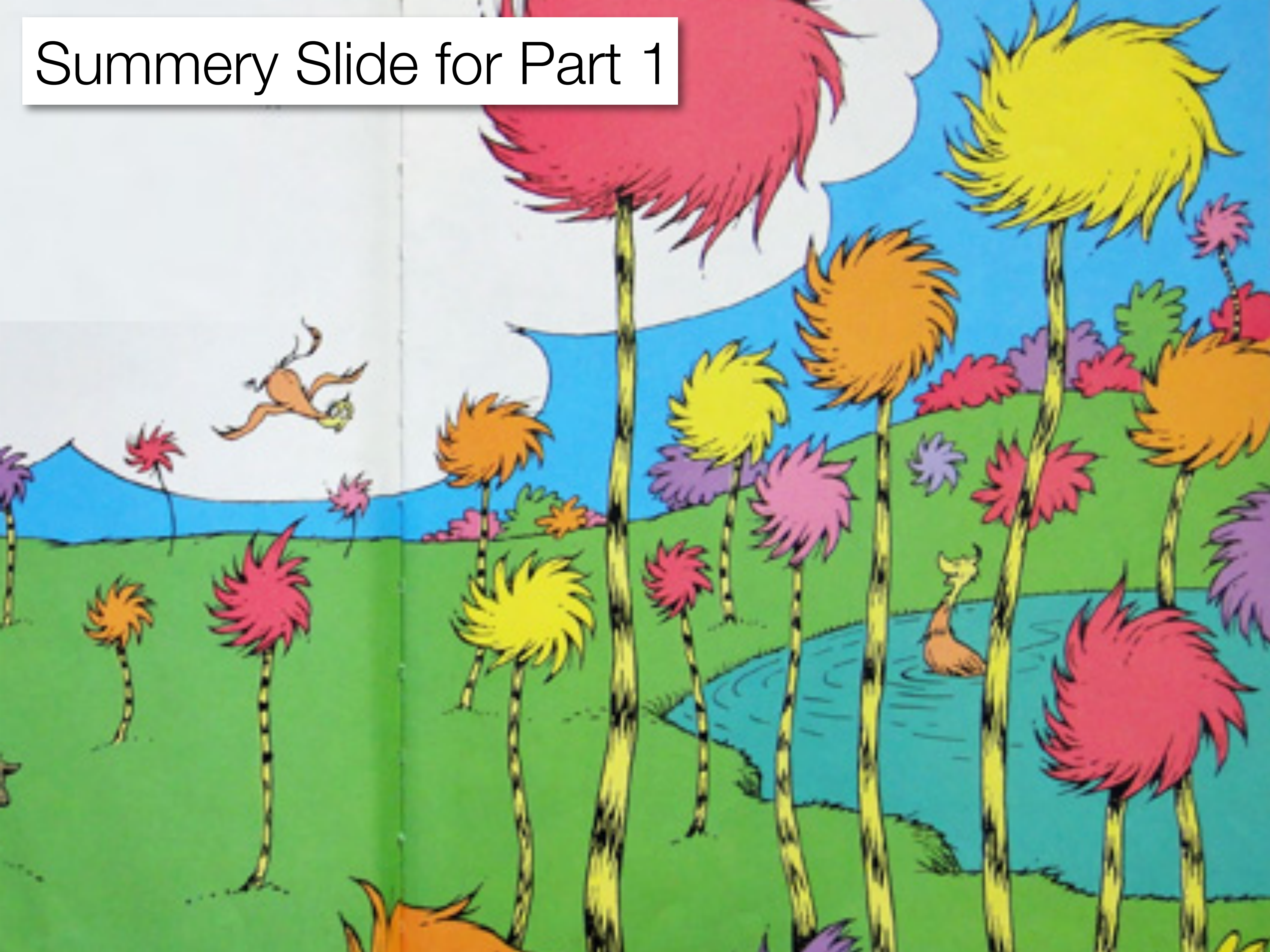
Exponential backoff and bursts

Exponential backoff may not recover from bursts for a time superpolynomial in the size of the burst. [Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]



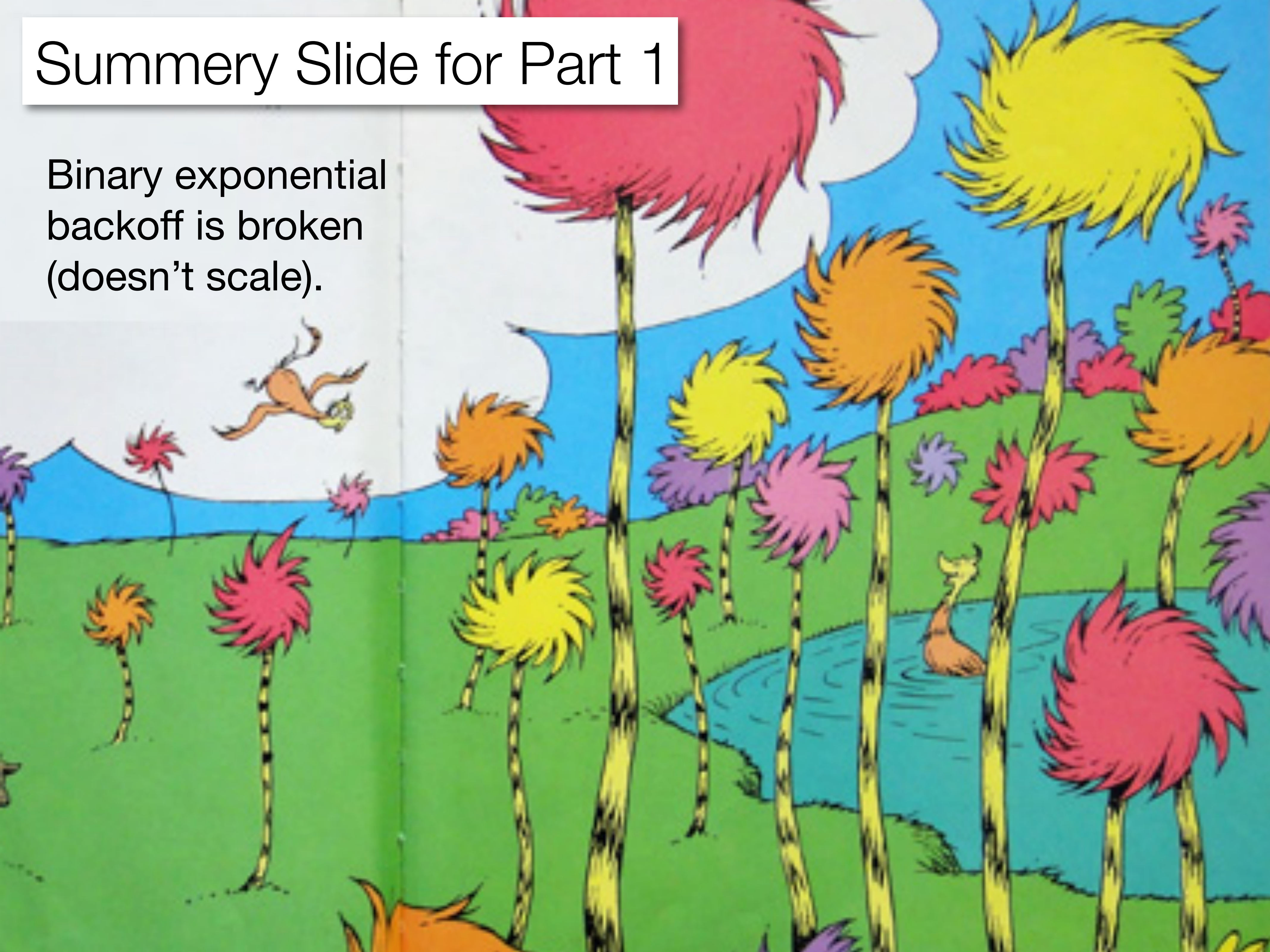
$$O(\log m) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\text{poly}(m)}$$

Summery Slide for Part 1



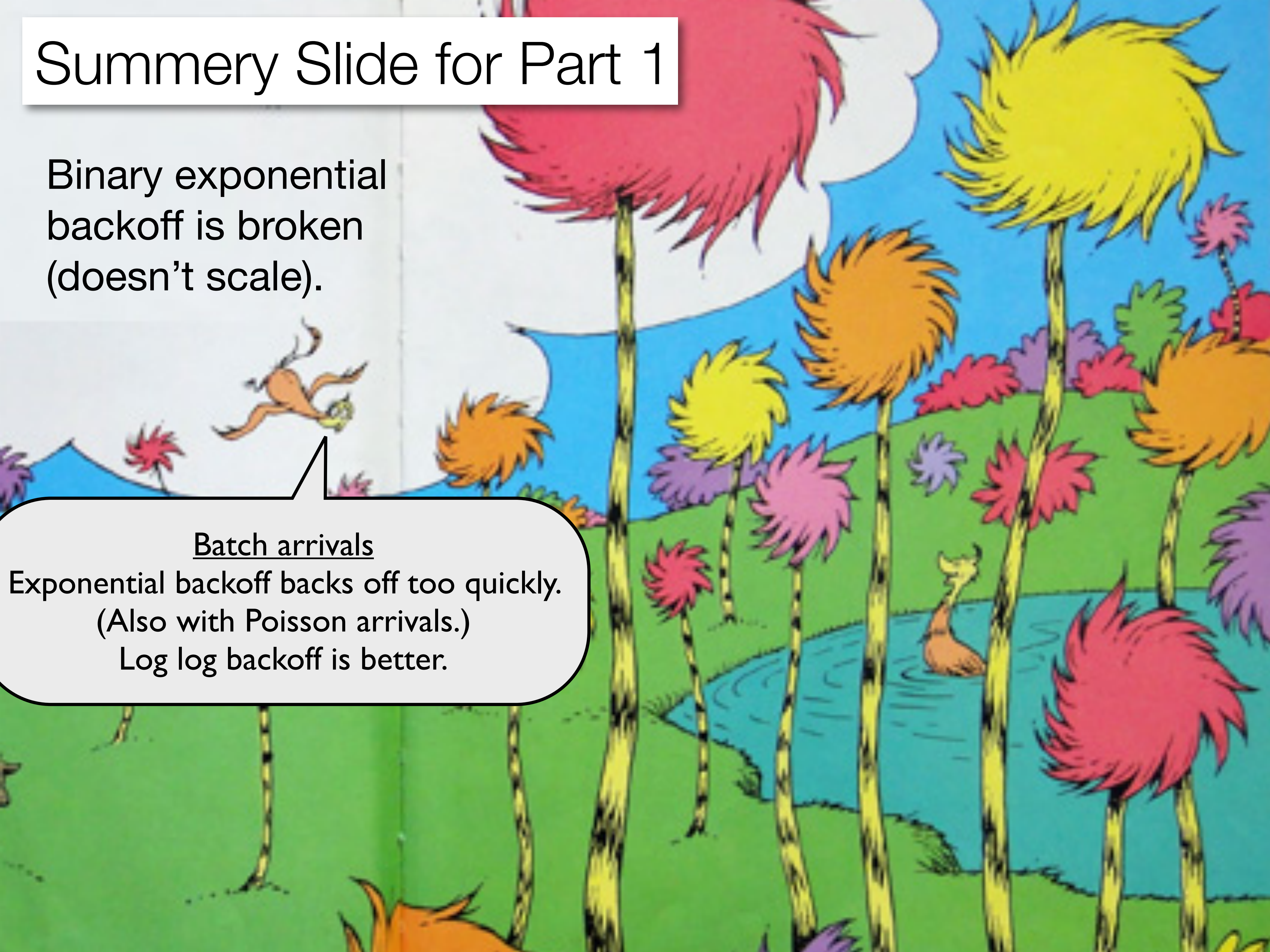
Summery Slide for Part 1

Binary exponential backoff is broken (doesn't scale).



Summery Slide for Part 1

Binary exponential backoff is broken (doesn't scale).



<u>Batch arrivals</u>

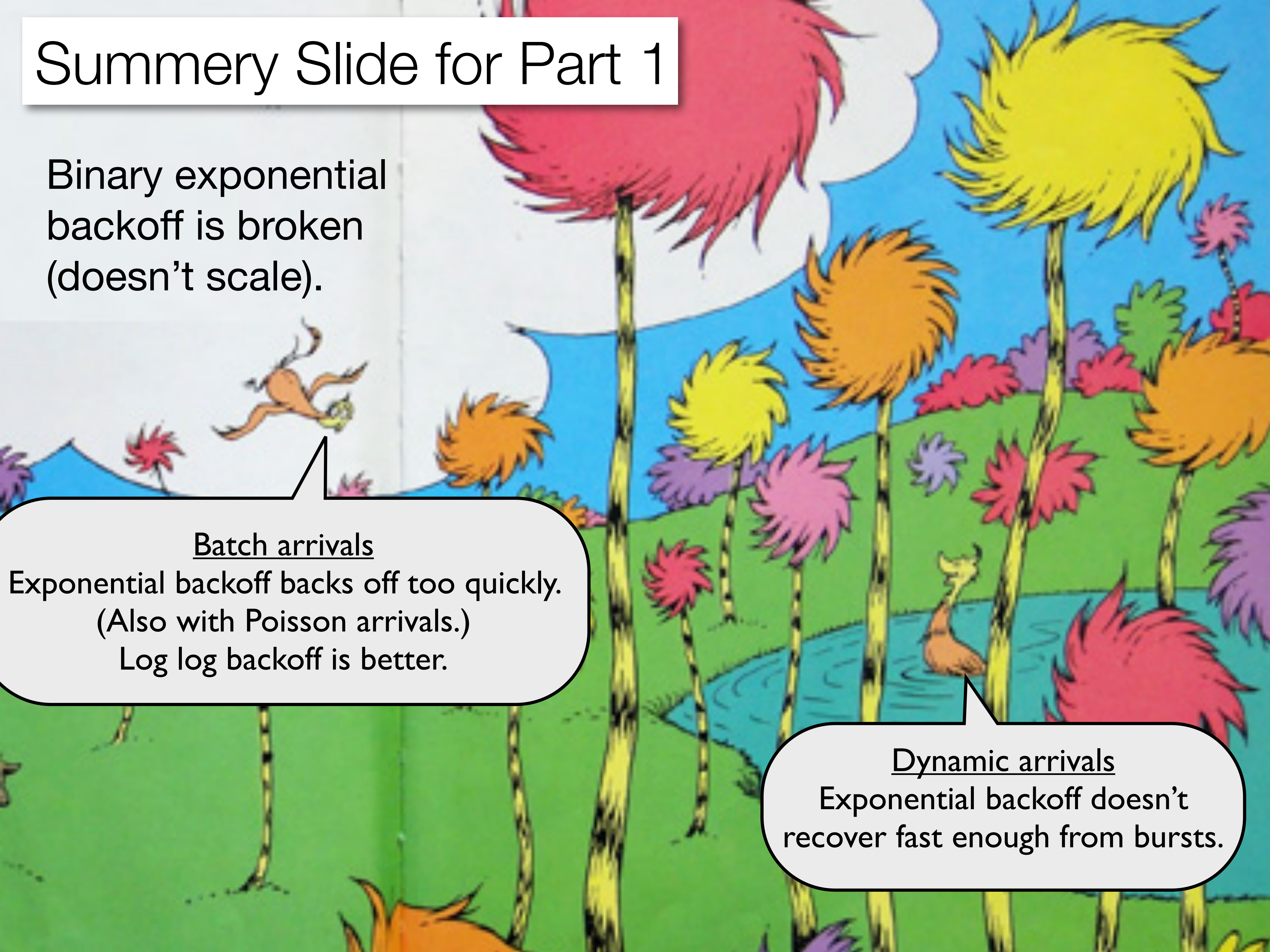
Exponential backoff backs off too quickly.

(Also with Poisson arrivals.)

Log log backoff is better.

Summery Slide for Part 1

Binary exponential backoff is broken (doesn't scale).



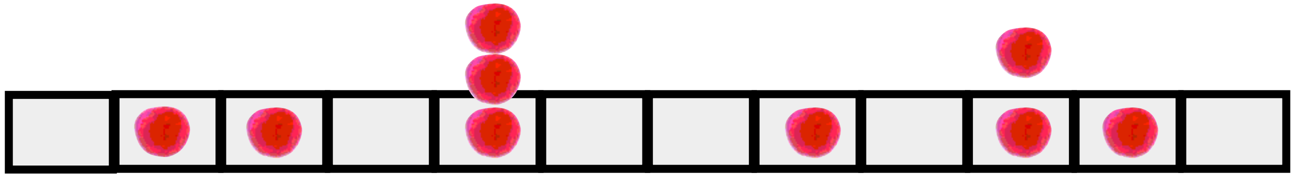
Part 2: TBD
Three Backoff Dilemmas

How to....

- maximize throughput,
- minimize # tries to access resource,
- achieve robustness.

1. Throughput

$$\frac{\#\text{ successful slots}}{\text{total number of slots}}$$



throughput = 4/12

(We'll need to generalize for dynamic arrivals.)

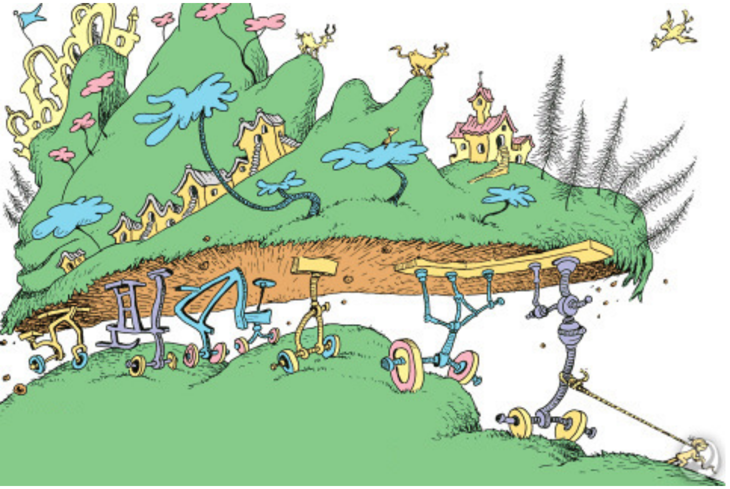
2. Minimize Effort / Attempts

Each broadcast (attempt to access resource) has a cost.

- In a wireless network, this cost is *energy*.
- In transactional memory, this cost is *processor cycles*.

Goal: Minimize the number of broadcasts.

- Exponential backoff: O(log *n*) on average. (But poor throughput.)
- Better: O($\log^2 n$) on average plus good throughput.



3. Cope with Failures / Disruption

Everything is unreliable

Broadcast channels fail

- wireless disruption
- adversarial jamming
- solar flares

Transactional memory fails

- guarantees are only best effort

Network links fails

- congestion
- router failures
- misconfiguration

3. Cope with Failures / Disruption

Failures can occur even without collisions.

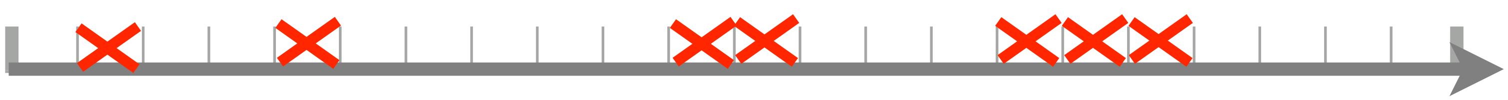
- In a wireless network: *noise* and/or *jamming*.
- In transactional memory: *best-effort hardware*

3. Cope with Failures / Disruption

Failures can occur even without collisions.

- In a wireless network: *noise* and/or *jamming*.
- In transactional memory: *best-effort hardware*

Model: adversary can block slots arbitrarily.

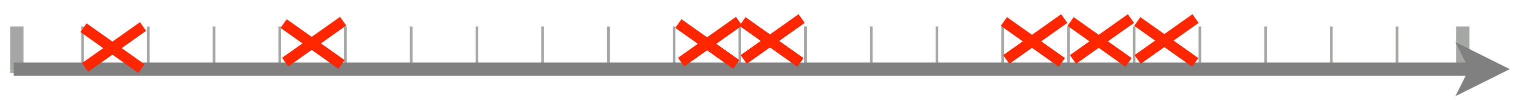


3. Cope with Failures / Disruption

Failures can occur even without collisions.

- In a wireless network: *noise* and/or *jamming*.
- In transactional memory: *best-effort hardware*

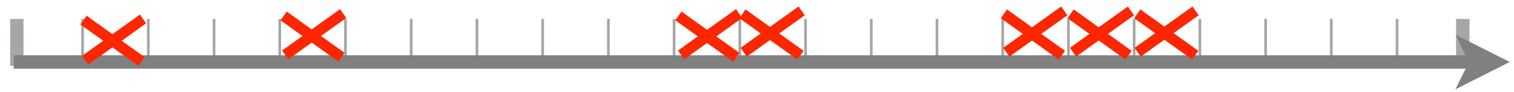
Model: adversary can block slots arbitrarily.



In any blocked slot:

- Every transmission attempt fails.
- Everyone senses that the slot is full.

3. Cope with Failures / Disruption



Goal: Constant throughput despite failures

- Waste at most a constant fraction of the slots

Examples

- Jamming Resistant MAC Protocols [Awerbuch, Richa, Scheideler ’08] [Richa, Scheideler, Schmid, Zhang ‘10] [Richa, Scheideler, Schmid, Zhang 11] [Richa, Scheideler, Schmid, Zhang ’12]
  - Adversary can jam O(1) fraction of the slots
  - fixed-station versus packet centric
- Resource-competitive analysis [King, Saia, Young ’11] [Gilbert, Young ’12] [Gilbert, King, Pettie, Porat, Saia, Young’14]
  - adversary can jam arbitrary, but there is a cost for this jamming.

Part 3: how to fix binary exponential backoff

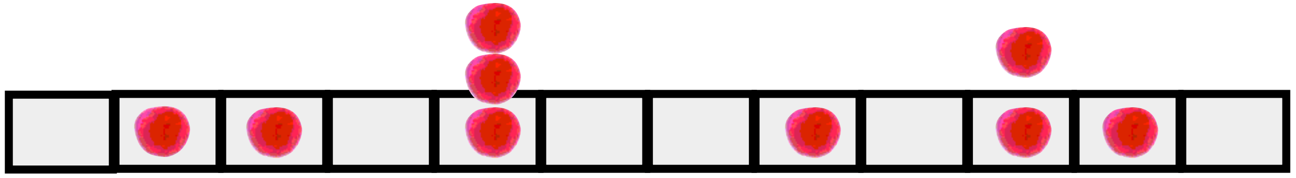
Analysis in two settings:

- batch (a single burst)
- dynamic arrivals

Batch arrivals

TBD

maximize throughput
minimize effort
achieve robustness



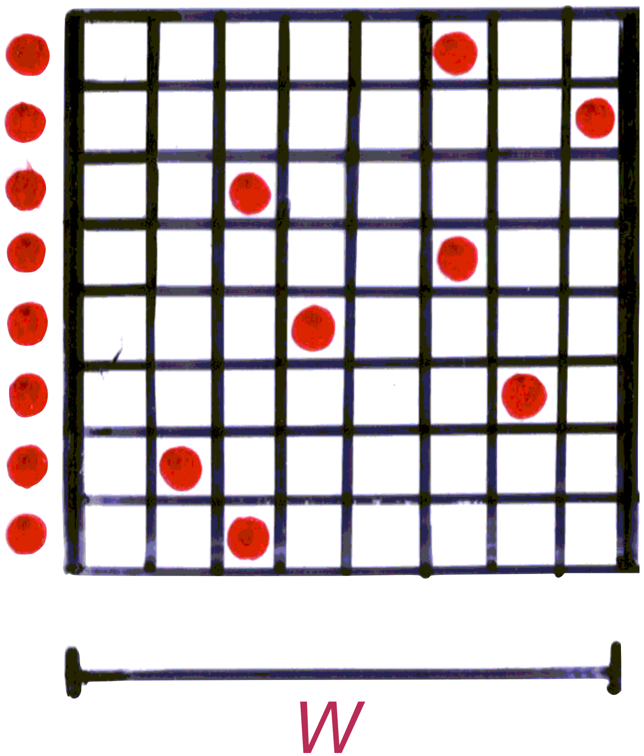
throughput = 4/12

Constant throughput for batches

Claim: When $W=\Theta(n)$, there are $\Theta(n)$ successes w.h.p..

Upshot: We can reduce W by a constant factor.

Corollary: All packets transmit in $\Theta(n)$ w.h.p..

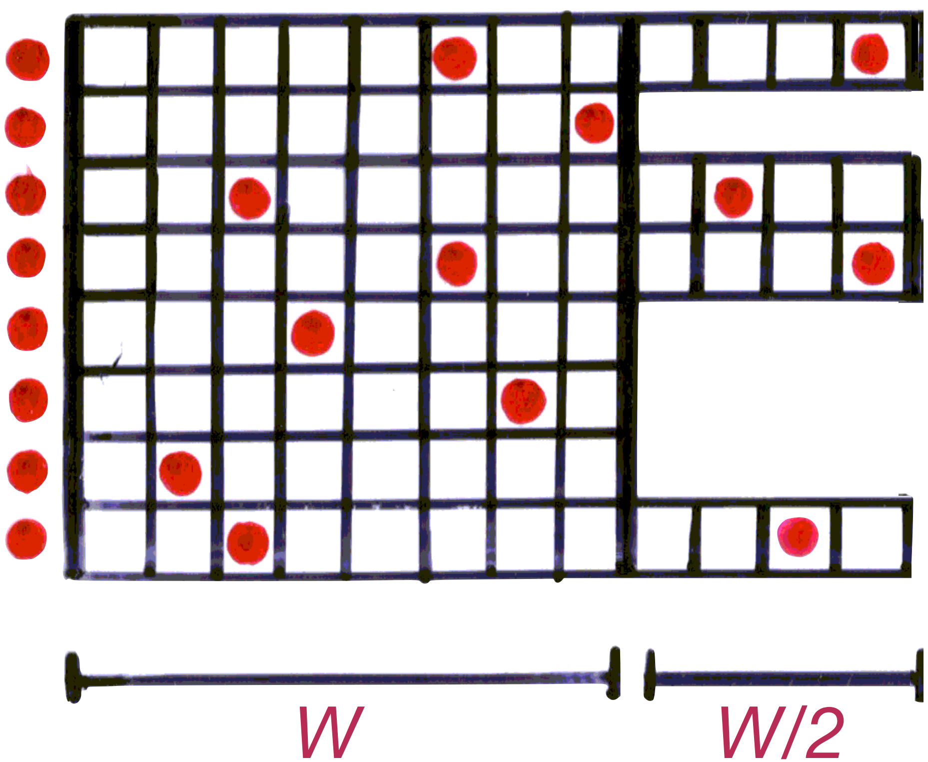


Constant throughput for batches

Claim: When $W=\Theta(n)$, there are $\Theta(n)$ successes w.h.p..

Upshot: We can reduce W by a constant factor.

Corollary: All packets transmit in $\Theta(n)$ w.h.p..

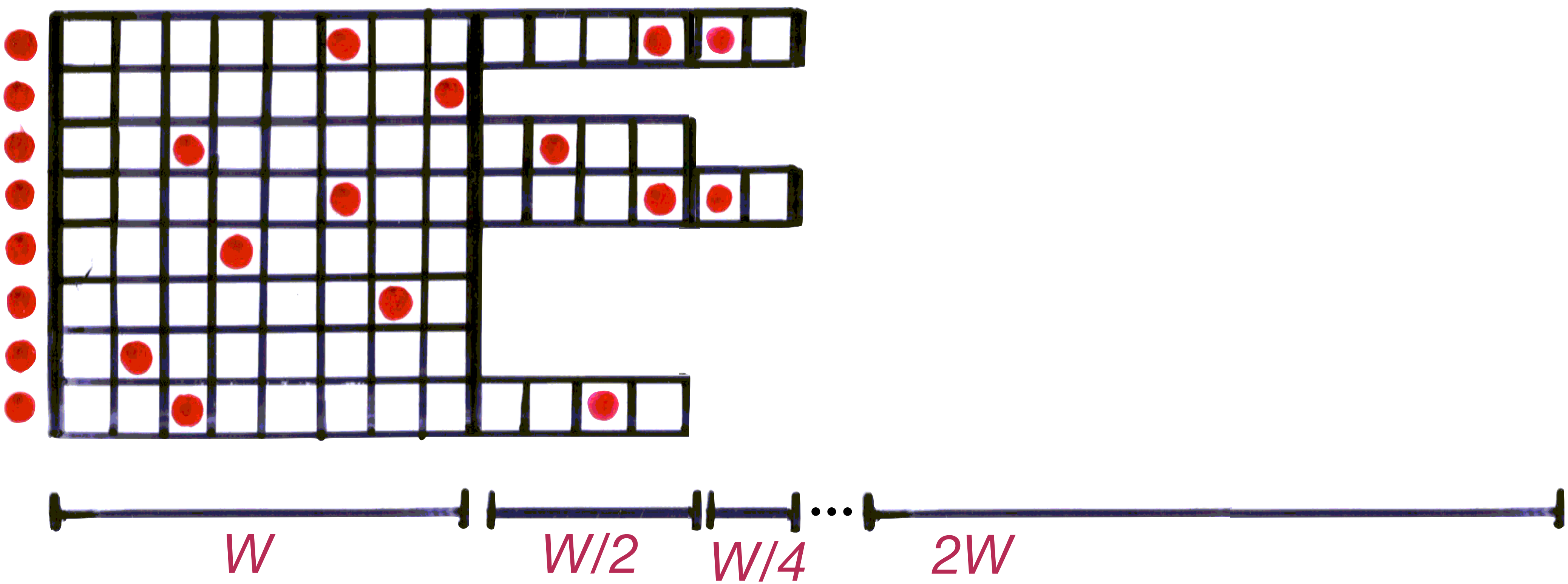


Constant throughput for batches

Claim: When $W=\Theta(n)$, there are $\Theta(n)$ successes w.h.p..

Upshot: We can reduce W by a constant factor.

Corollary: All packets transmit in $\Theta(n)$ w.h.p..



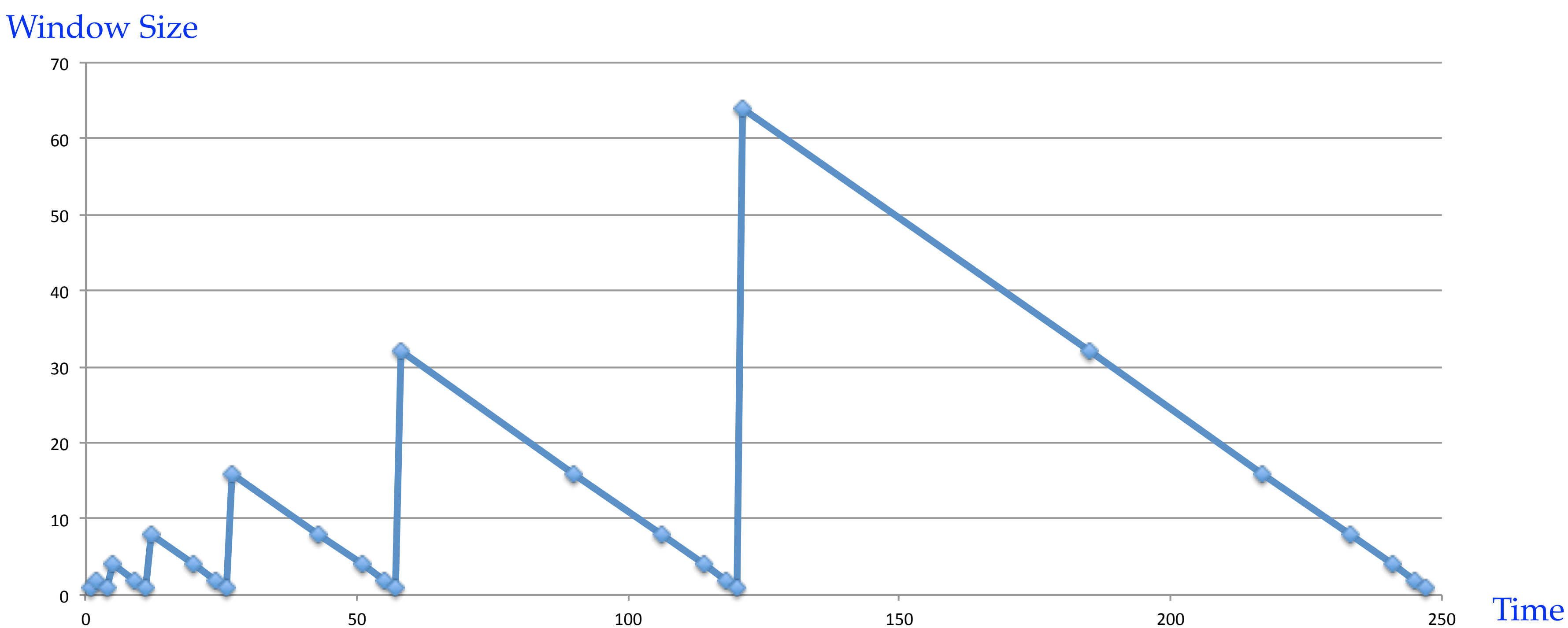
Sawtooth backoff

[Greenberg and Leiserson '89]
[Gereb-Graus and Tsantilas '92]
[Bender, Farach-Colton, He, Kuszmaul, Leiserson '05]

Guess a value of $W = n$.

Back on with window size $W/2$, $W/4$, $W/8$, …

Back off with $W = 2n$.



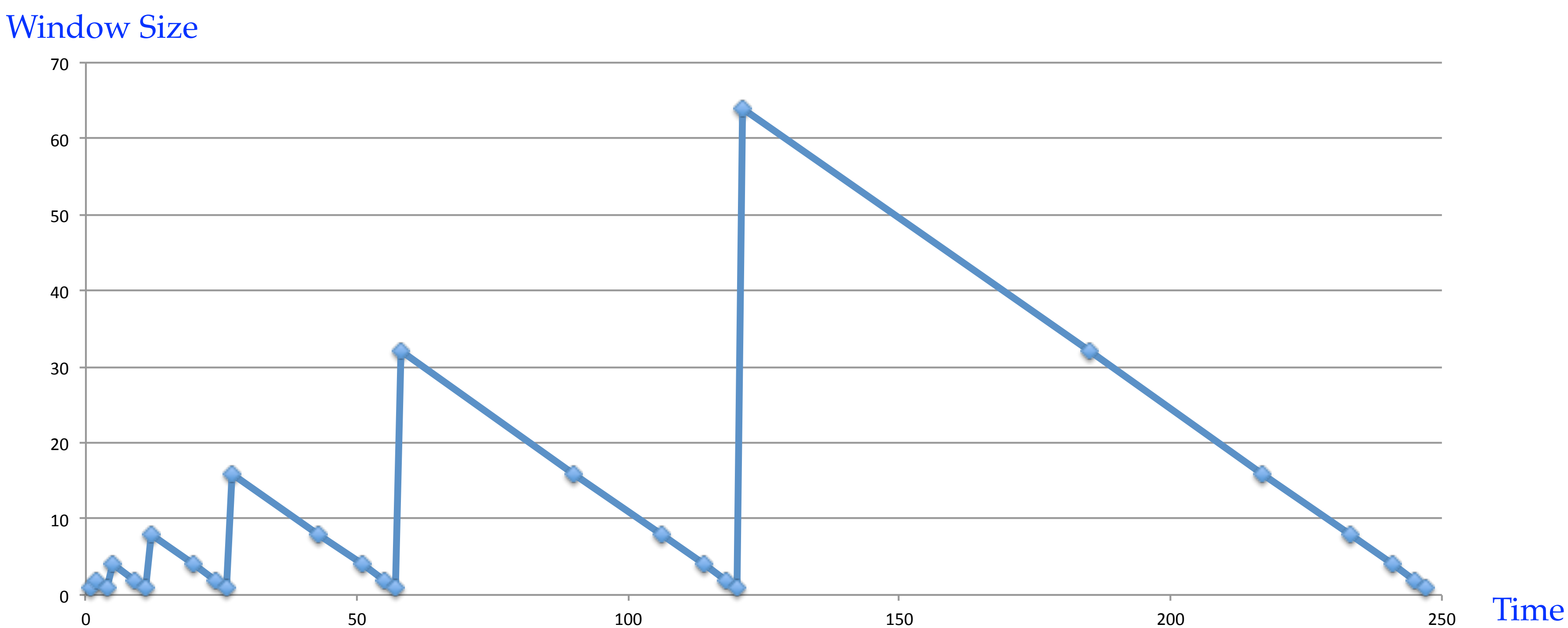
Sawtooth backoff

[Greenberg and Leiserson '89]
[Gereb-Graus and Tsantilas '92]
[Bender, Farach-Colton, He, Kuszmaul, Leiserson '05]

Theorem: For n packet that arrive at time 0, w.h.p., all packets transmit after

O(n) time ⇒ O(1) throughput

O($\log^2 n$) attempts.



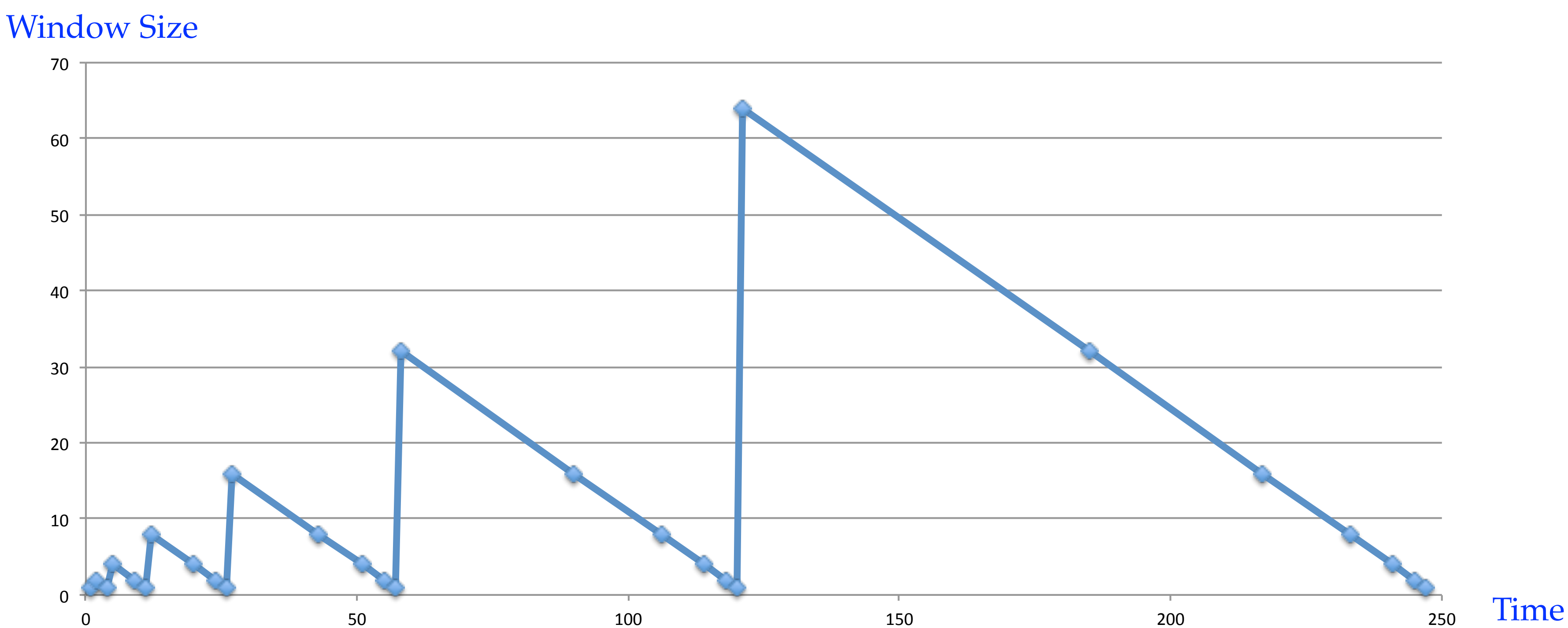
Sawtooth backoff

[Greenberg and Leiserson '89]
[Gereb-Graus and Tsantilas '92]
[Bender, Farach-Colton, He, Kuszmaul, Leiserson '05]

Theorem: For n packet that arrive at time 0, w.h.p., all packets transmit after

$O(n)$ time $\Rightarrow$ $O(1)$ throughput

$O(\log^2 n)$ attempts.

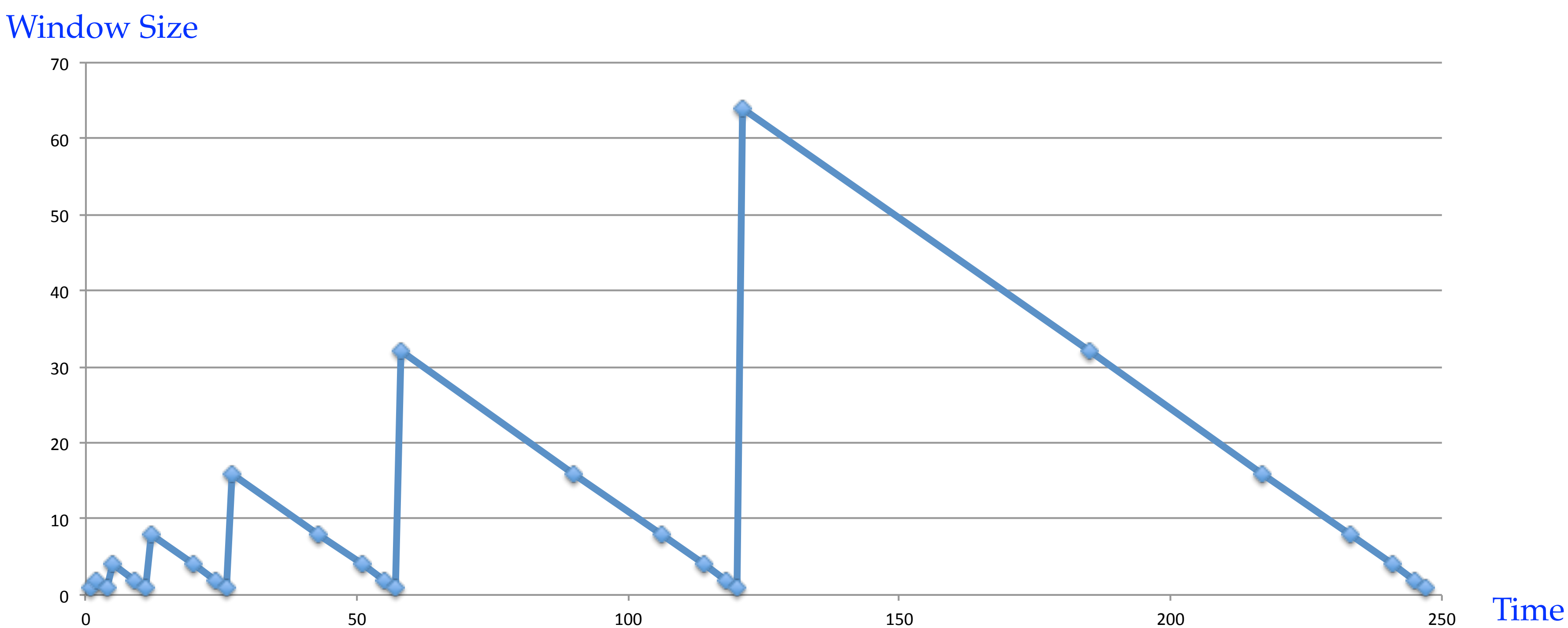


Sawtooth backoff

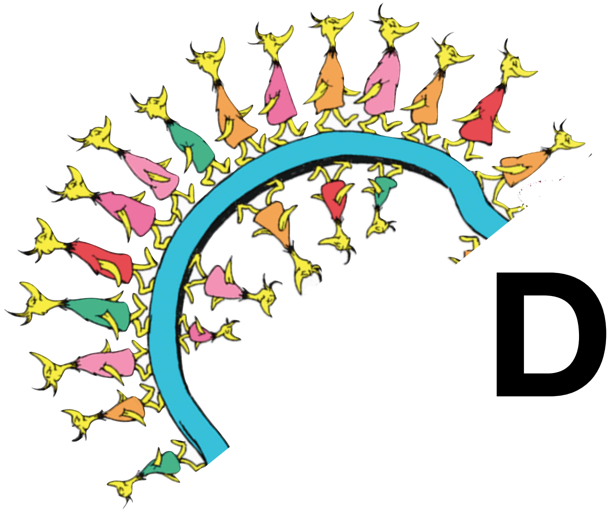
[Greenberg and Leiserson '89]
[Gereb-Graus and Tsantilas '92]
[Bender, Farach-Colton, He, Kuszmaul, Leiserson '05]

Theorem: For n packet that arrive at time 0, w.h.p., all packets transmit after

- $O(n)$ time $\Rightarrow$ $O(1)$ throughput
- $O(\log^2 n)$ attempts.



Robust to failures. (I'll state a theorem later.)
But lousy with dynamic arrivals.



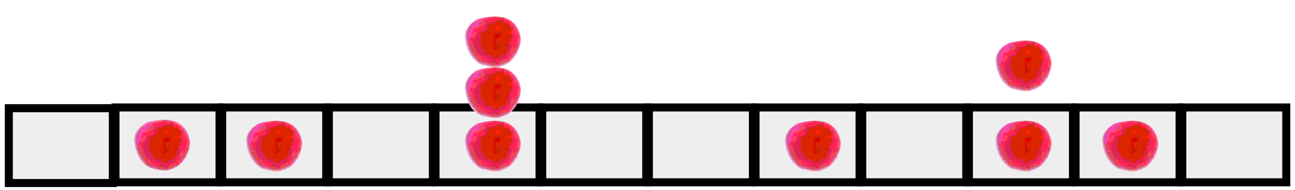
Dynamic arrivals

maximize throughput

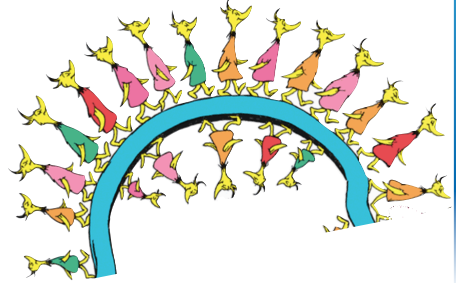
minimize effort

achieve robustness

[Bender, Fineman, GIlbert, Young 14]

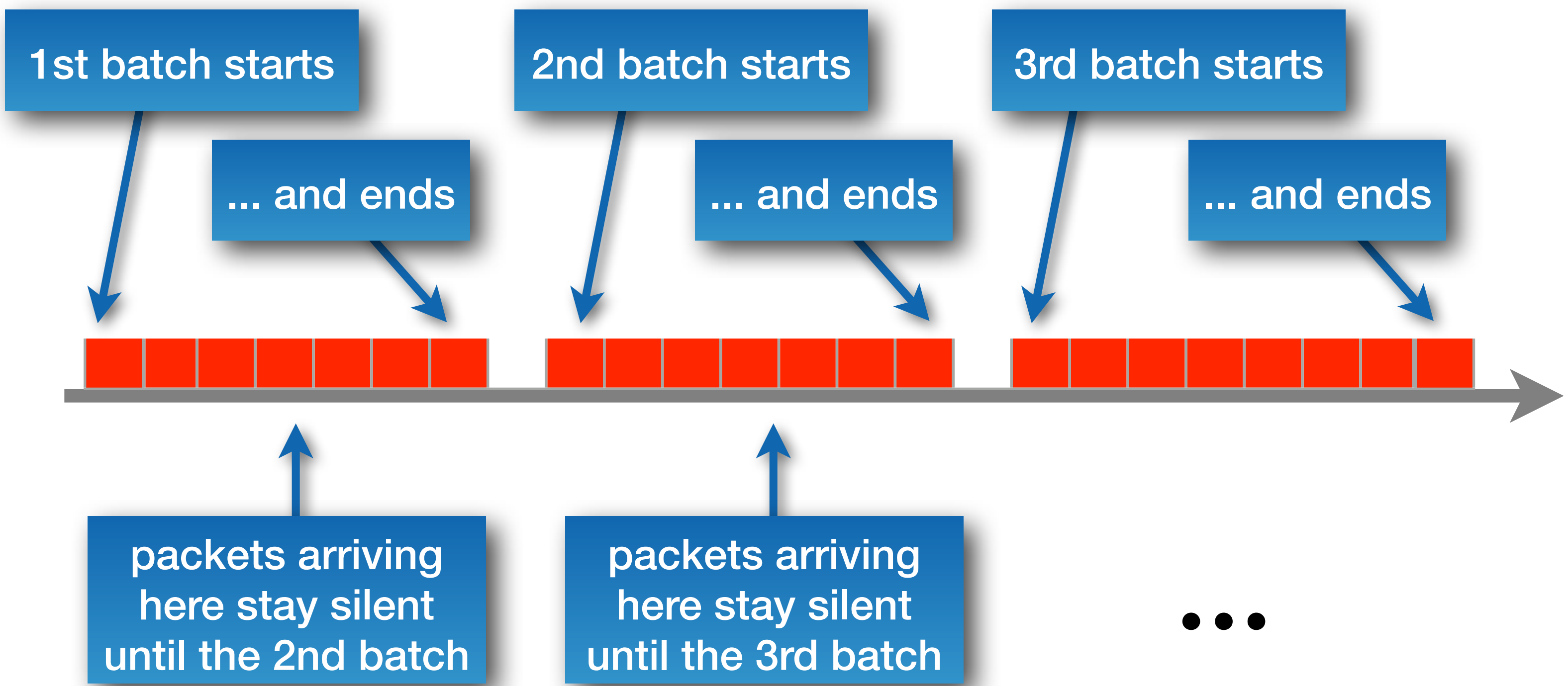


throughput = 4/12



Dynamic arrivals: synchronize into batches

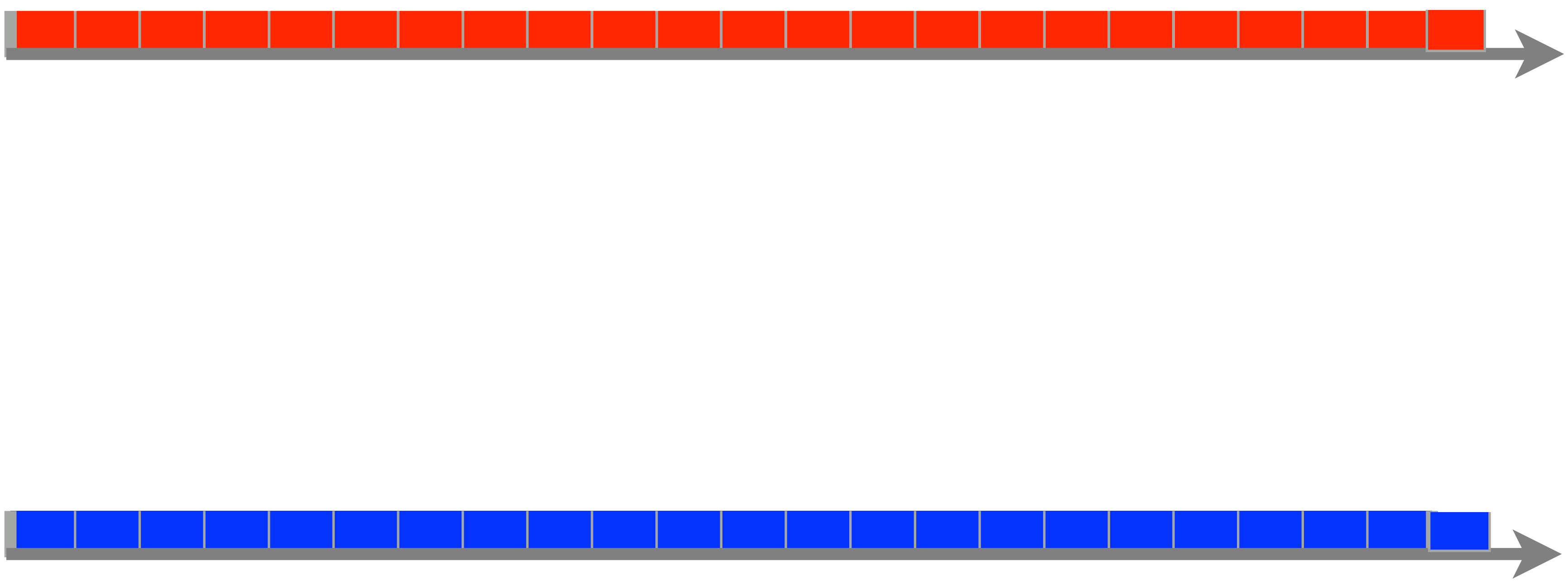
Group packets into synchronized batches.

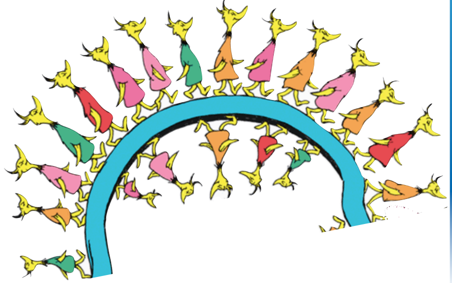


Use two channels (simulate on one)

Assume two channels.

We use the 2nd channel to synchronize into batches.





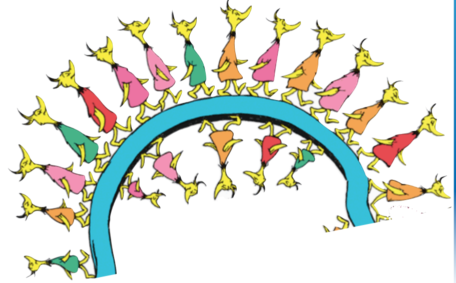
Use two channels (simulate on one)

Assume two channels.

We use the 2nd channel to synchronize into batches.

We can simulate two channels on one.

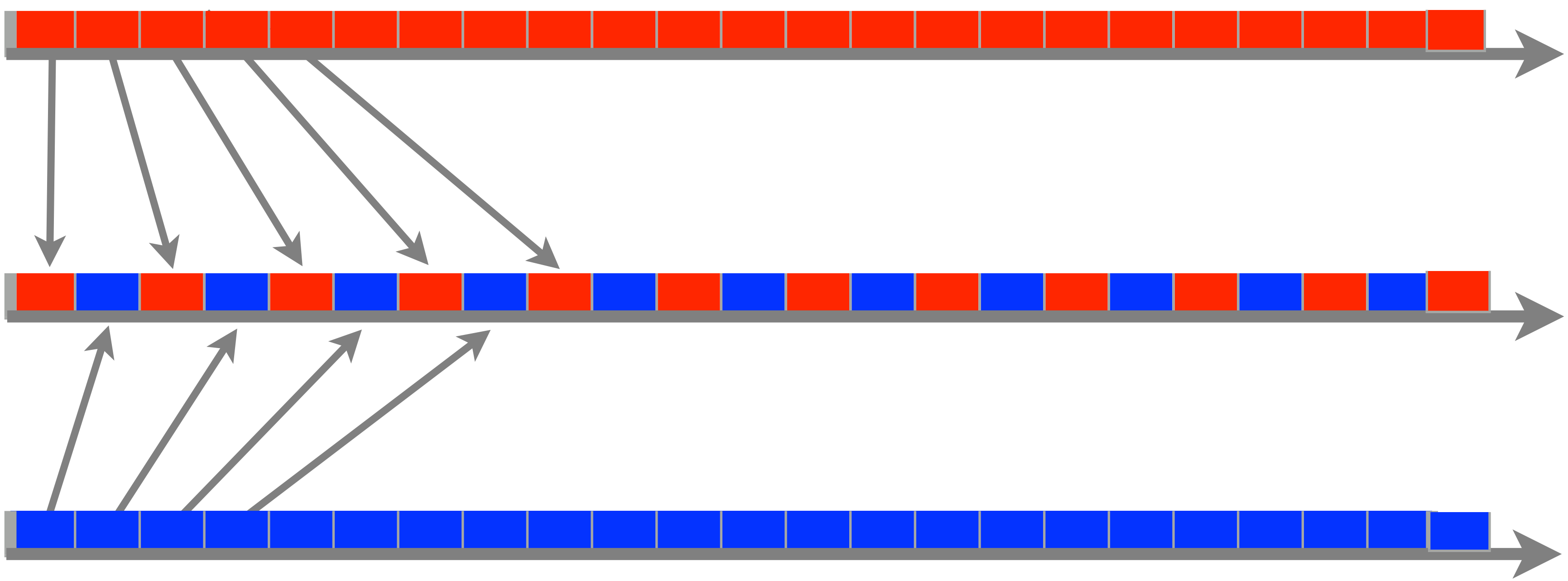
One assumption: *even/odd* round parity is known.



Use two channels (simulate on one)

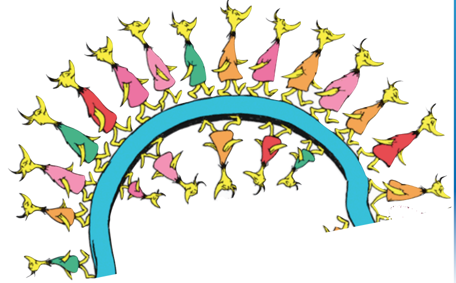
Assume two channels.

We use the 2nd channel to synchronize into batches.



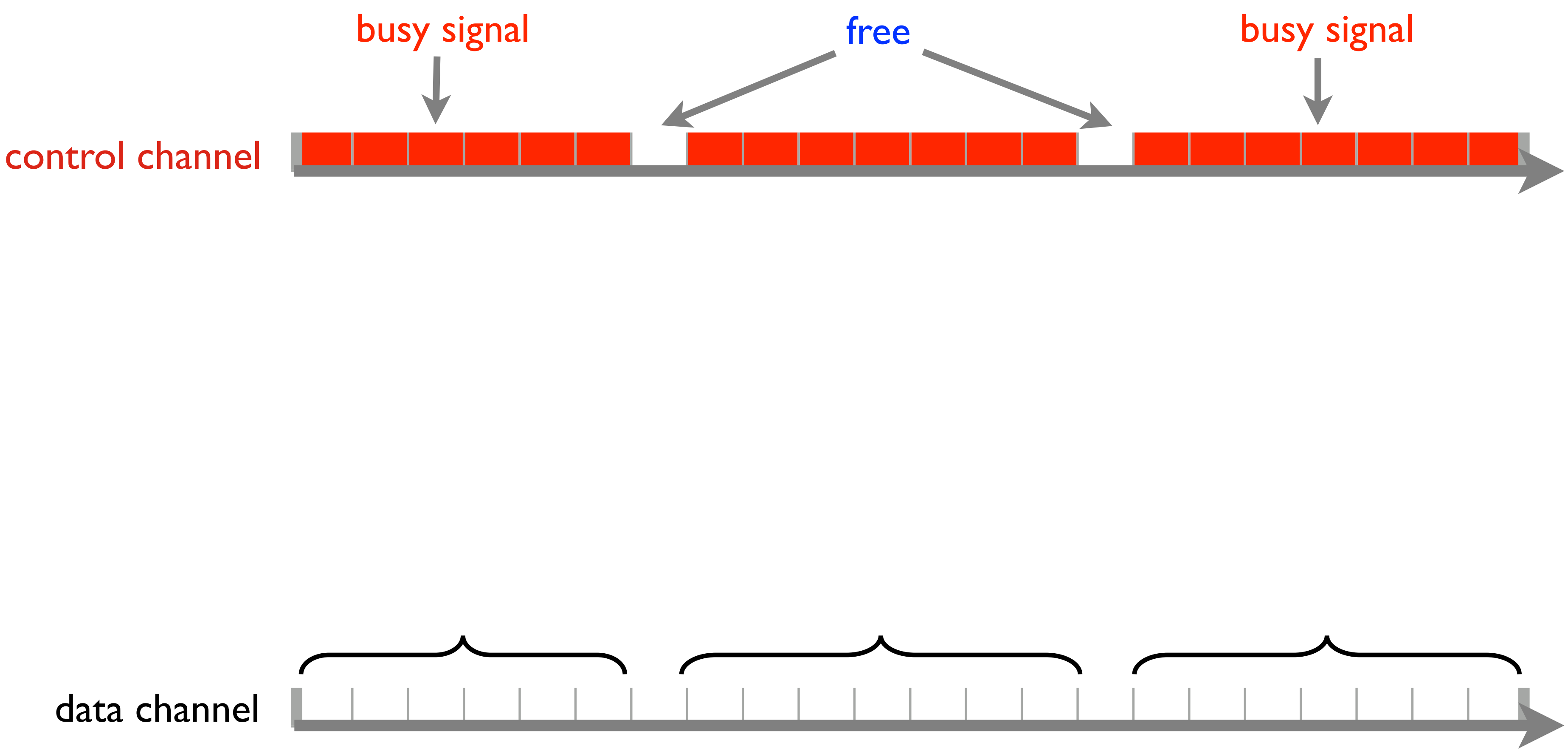
We can simulate two channels on one.

One assumption: *even/odd* round parity is known.



Synchronize batches using busy signal

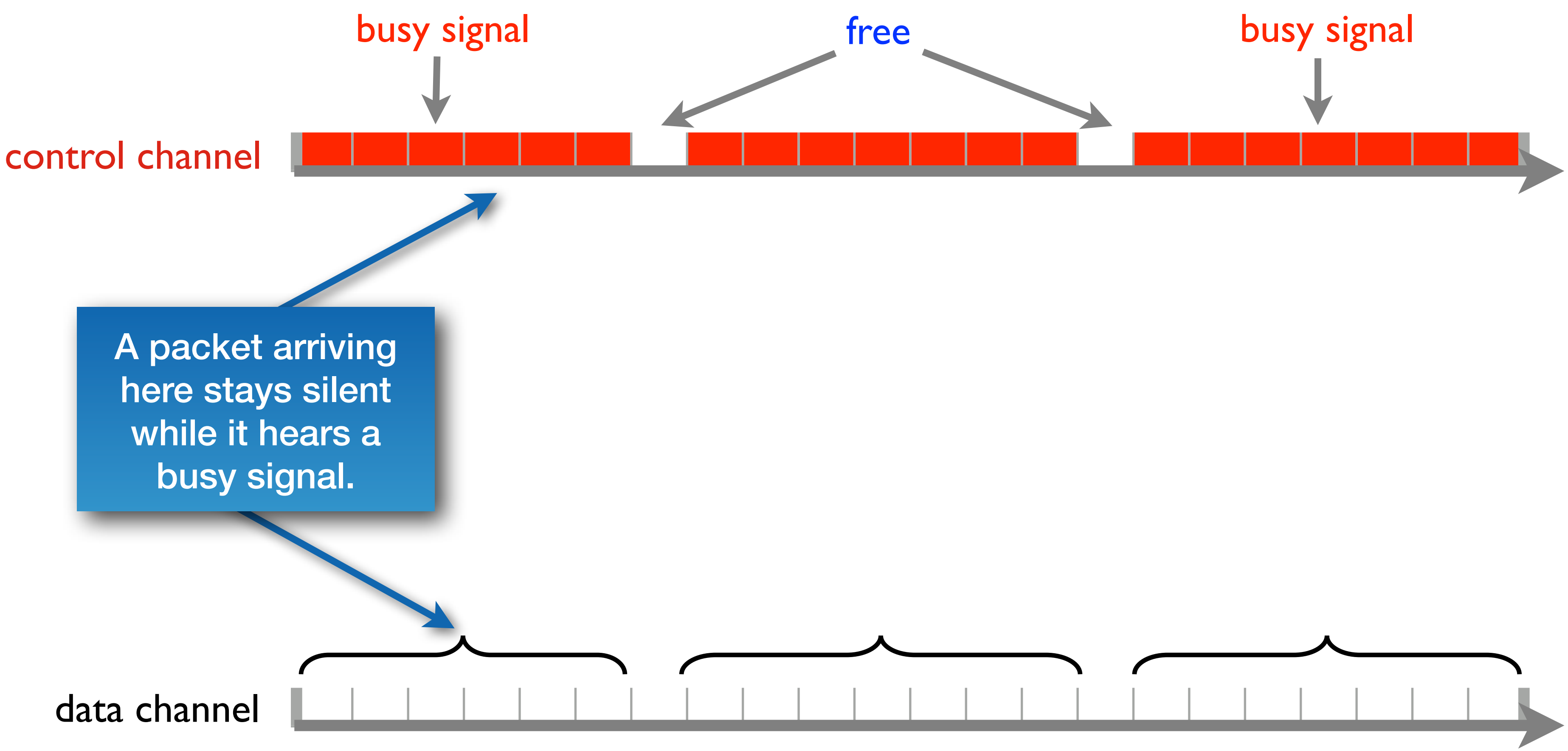
Control channel implements a busy signal [Wu and Li '88] [Haas and Deng '02] .



Data channel implements batches.

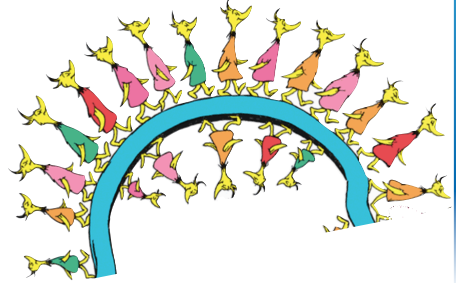
Synchronize batches using busy signal

Control channel implements a busy signal [Wu and Li '88] [Haas and Deng '02] .

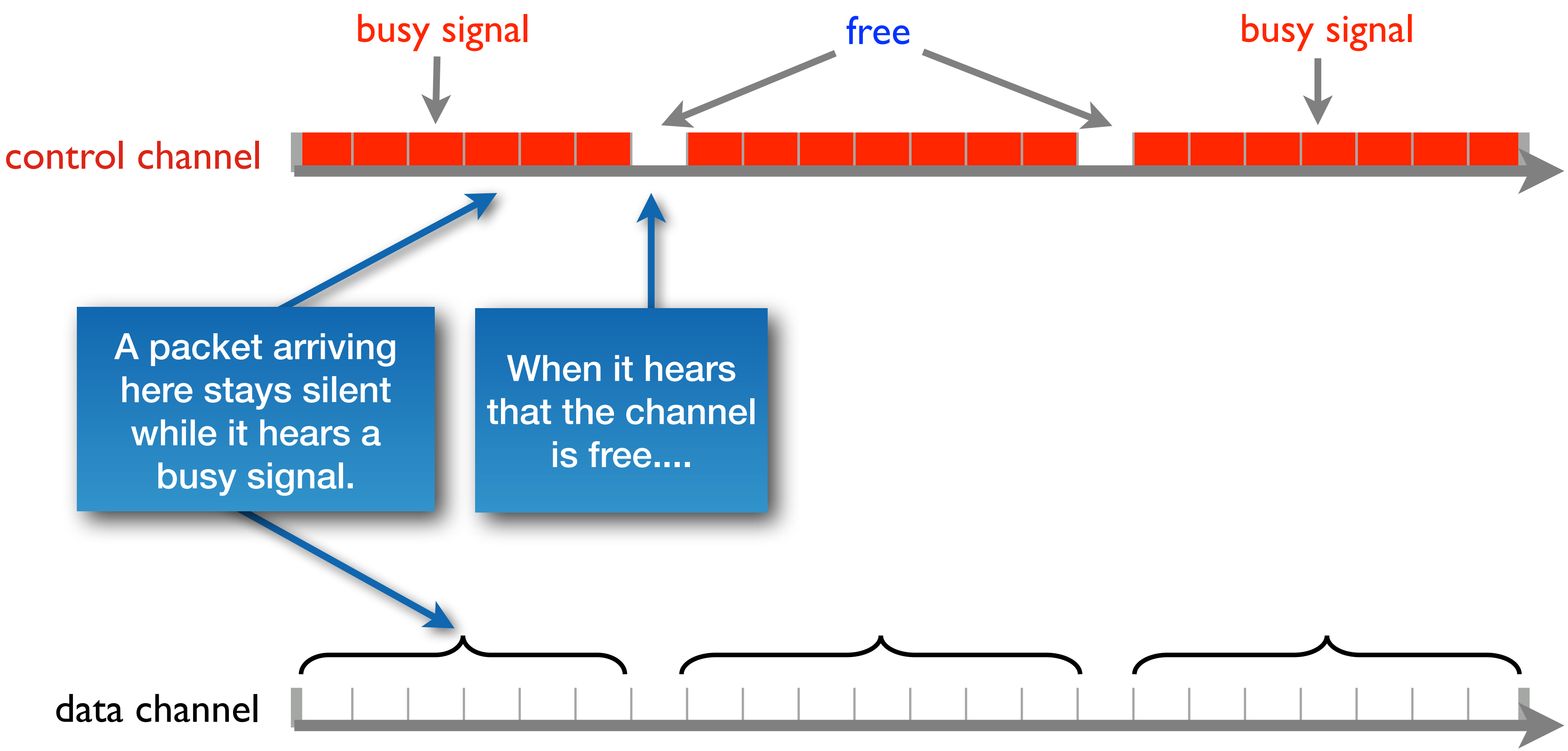


Data channel implements batches.

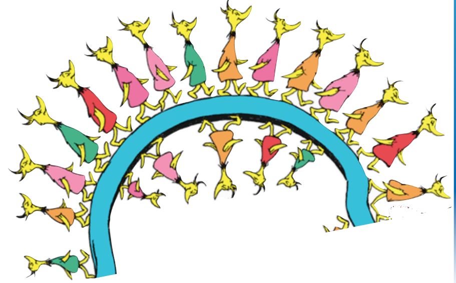
Synchronize batches using busy signal



Control channel implements a busy signal [Wu and Li '88] [Haas and Deng '02] .

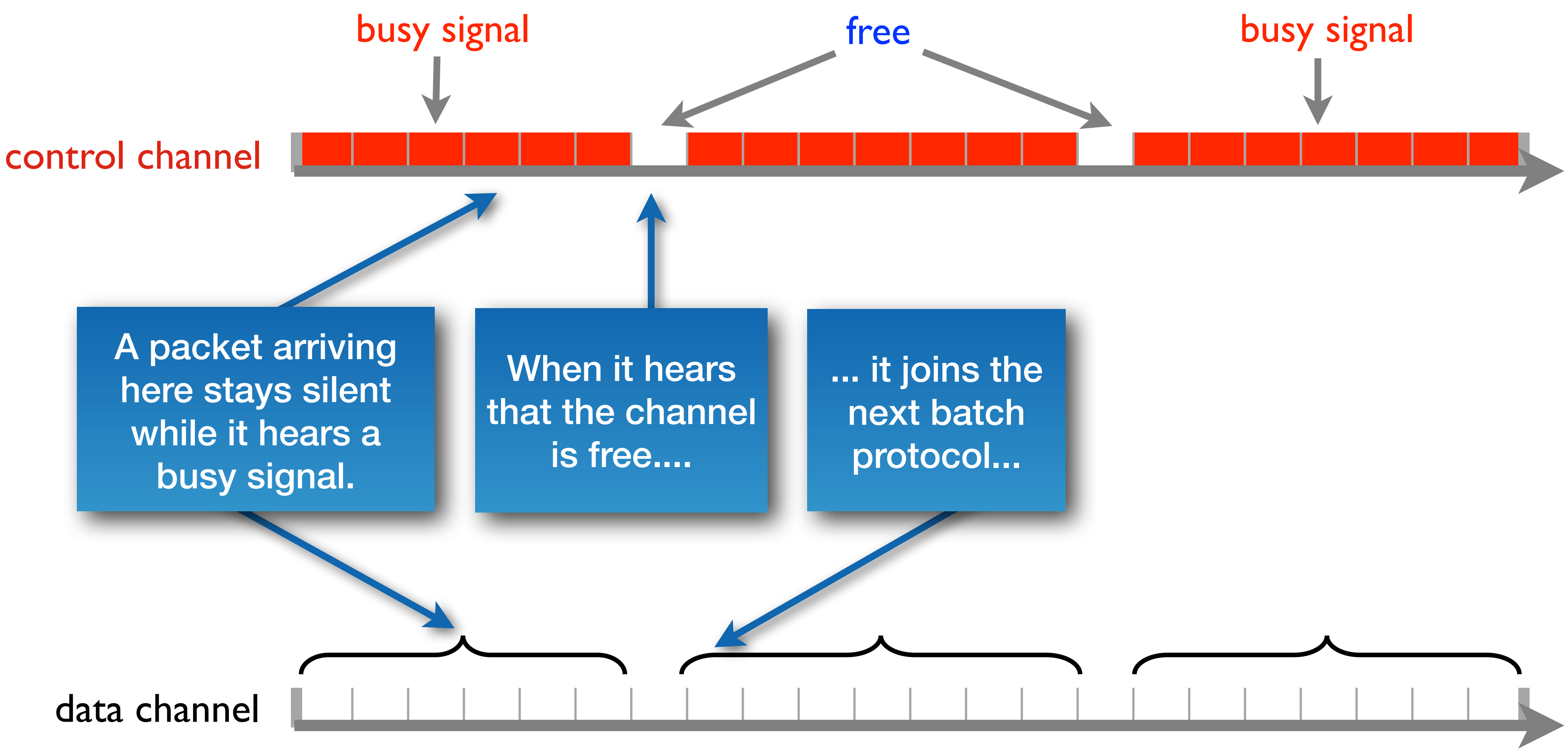


Data channel implements batches.

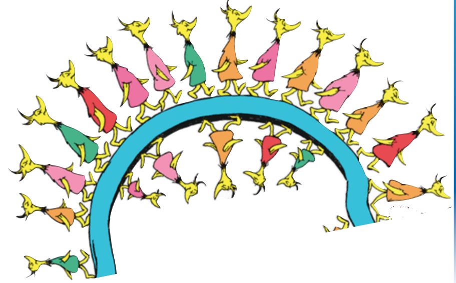


Synchronize batches using busy signal

Control channel implements a busy signal [Wu and Li ’88] [Haas and Deng ’02] .

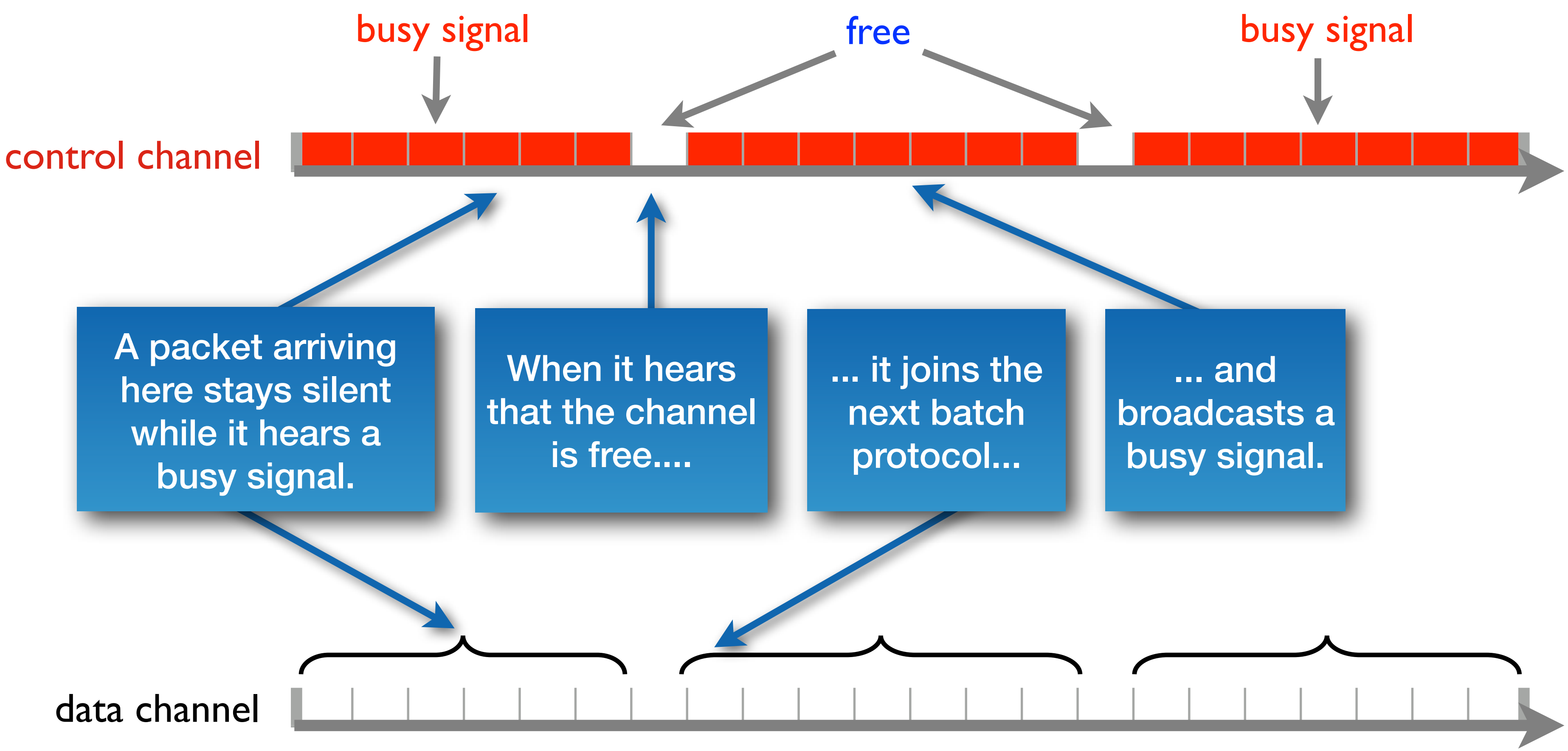


Data channel implements batches.

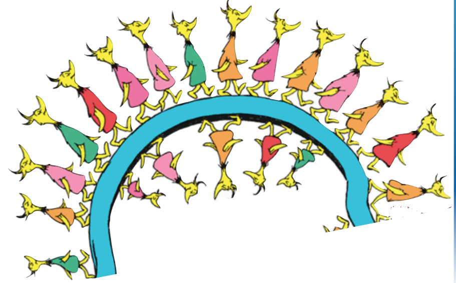


Synchronize batches using busy signal

Control channel implements a busy signal [Wu and Li ’88] [Haas and Deng ’02] .



Data channel implements batches.



Protocol on one channel

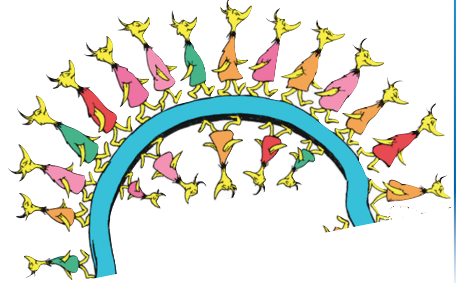
[Bender, Fineman, Gilbert, Young 14]

Wait until two consecutive "silent" rounds.

Set round counter to 0:

- **In odd rounds: broadcast (simulate *control channel*).**
- **In even rounds: run Sawtooth backoff (simulate *data channel*).**

Theorem: For *n* requests that arrive dynamically, Synchronized Sawtooth achieves $\Theta(1)$ throughput, w.h.p.



Protocol on one channel

[Bender, Fineman, GIlbert, Young 14]

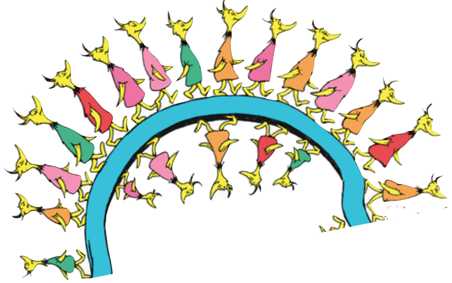
Wait until two consecutive "silent" rounds.

Set round counter to 0:

- **In odd rounds:** broadcast (simulate *control channel*).
- **In even rounds:** run Sawtooth backoff (simulate *data channel*).

Packets broadcast every other round.
O(n) attempts is expensive!

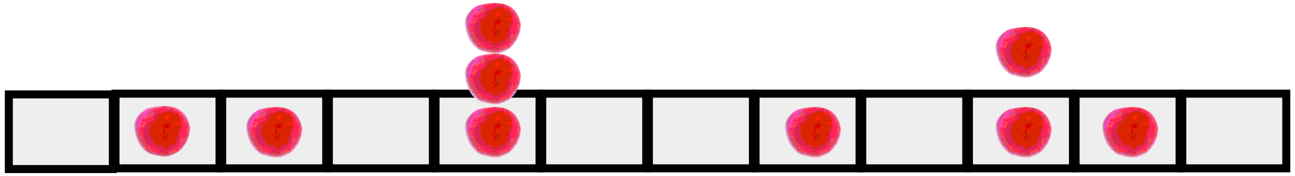
Theorem: For n requests that arrive dynamically, Synchronized Sawtooth achieves $\Theta(1)$ throughput, w.h.p.



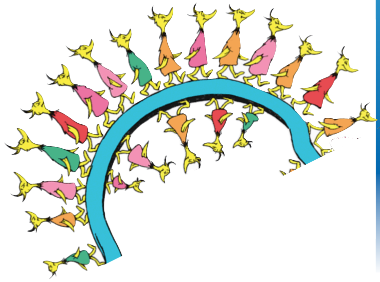
Dynamic arrivals

maximize throughput
minimize effort
achieve robustness

[Bender, Fineman, GIlbert, Young 14]

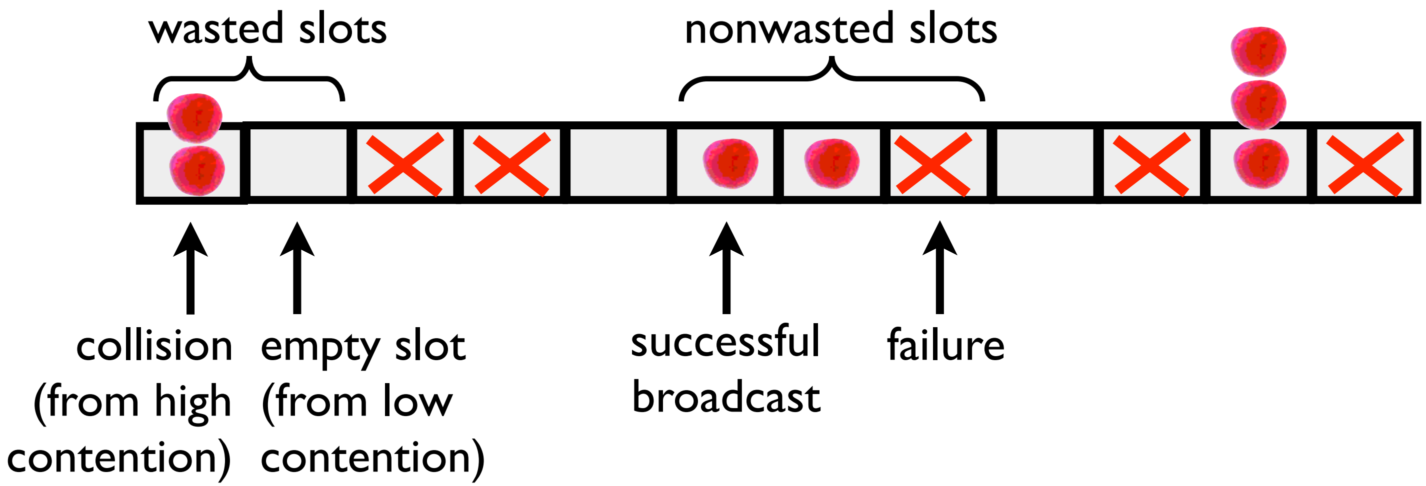


throughput = 4/12

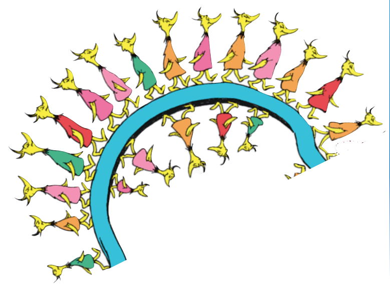


Throughput in the presence of failures

Constant throughput = waste at most O(1) fraction of slots.



(Recall: contention = sum of broadcast probabilities.)



Resolving TBD

[Bender, Fineman, GIlbert, Young 14]

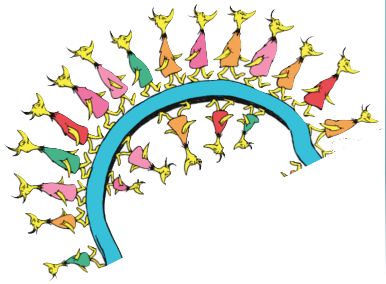
Theorem (for finite case):

Let *f* be the number of failed slots.

Let *n* be the number of (adversarially scheduled) packets.

We can achieve

- $\Theta(1)$ throughput in expectation,
  i.e., algorithm runs in time $O(n+f)$.
- $O(\log^2(n+f))$ broadcasts in expectation.



Resolving TBD

[Bender, Fineman, GIlbert, Young 14]

Theorem (for finite case):

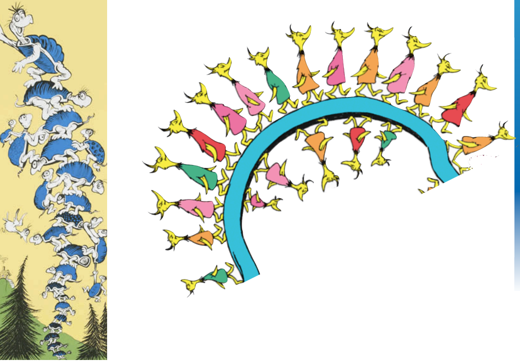
Let f be the number of failed slots.

Let n be the number of (adversarially scheduled) packets.

We can achieve

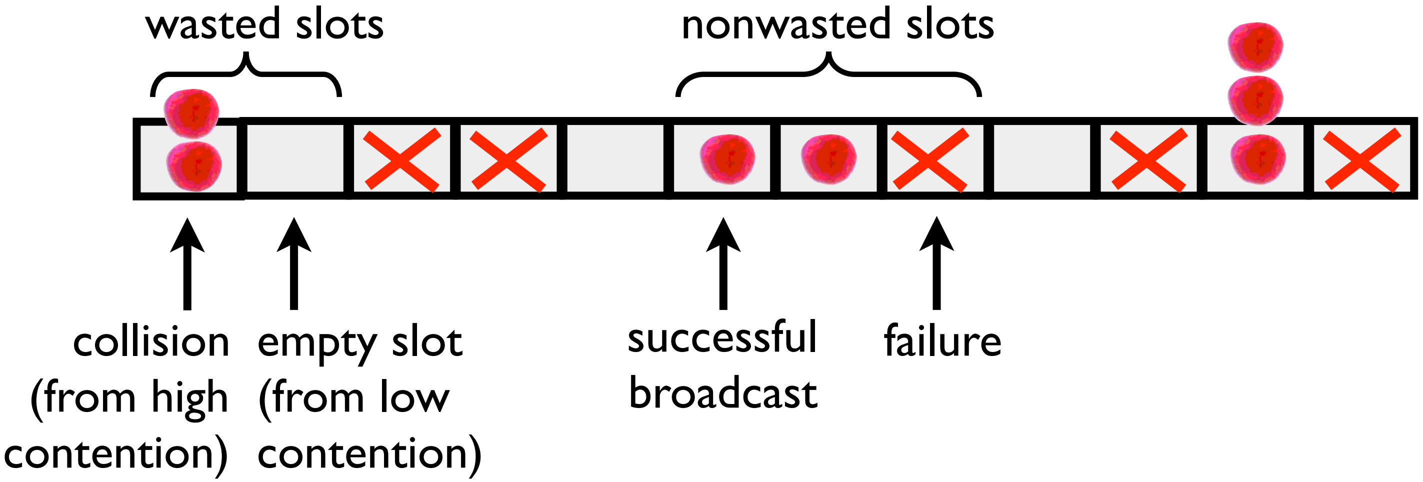
- $\Theta(1)$ throughput in expectation,
  i.e., algorithm runs in time $O(n+f)$.
- $O(\log^2(n+f))$ broadcasts in expectation.

There's a similar theorem for the infinite case.



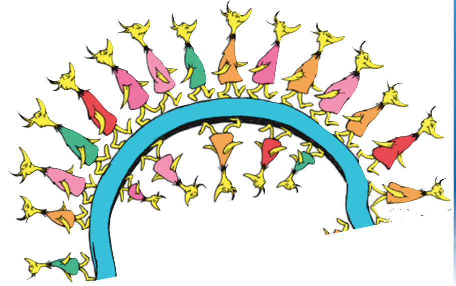
It's all about contention

Constant throughput = waste O(1) fraction of slots.



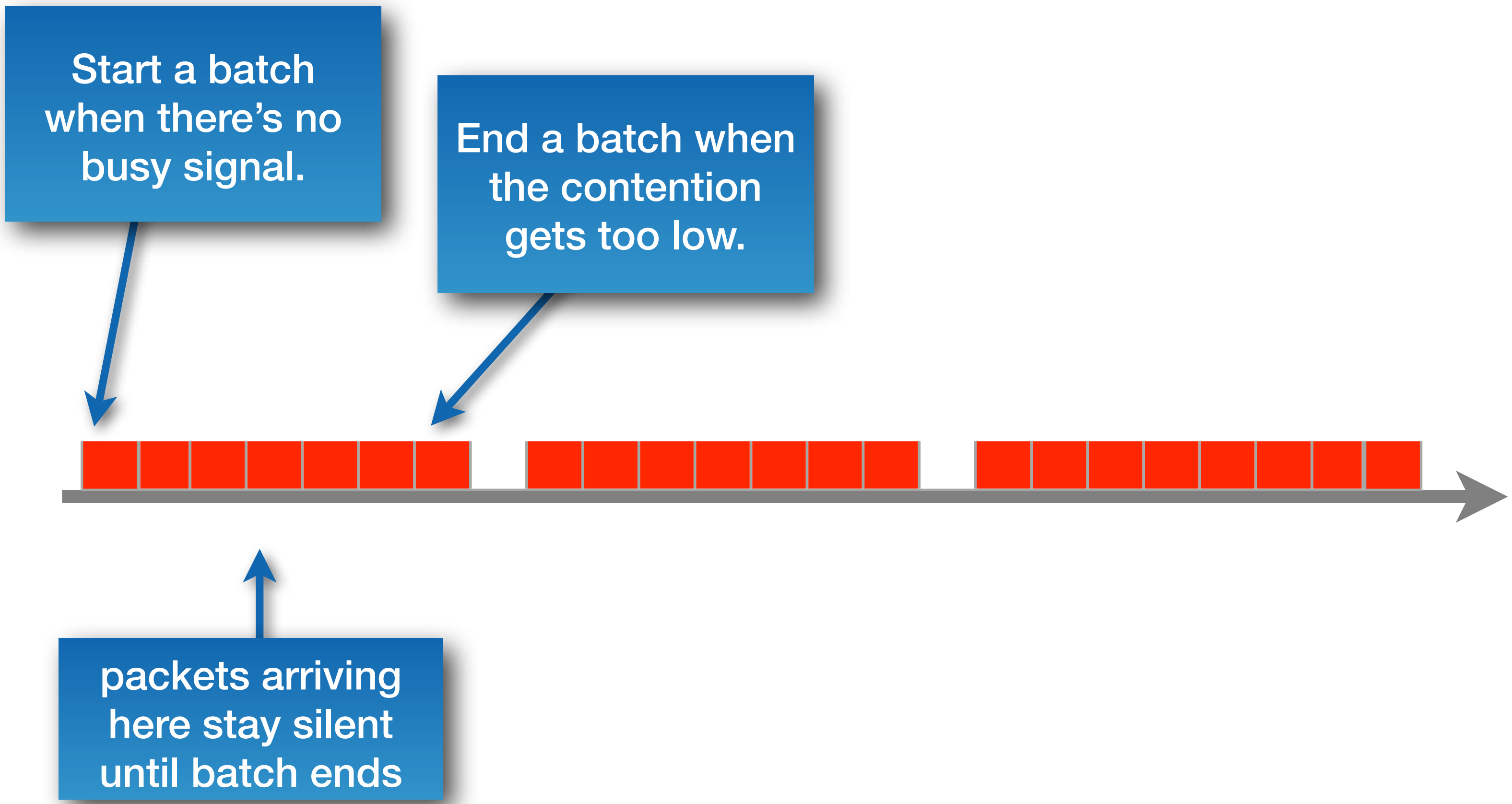
Goal: achieve Θ(1) contention on a constant fraction of all slots.

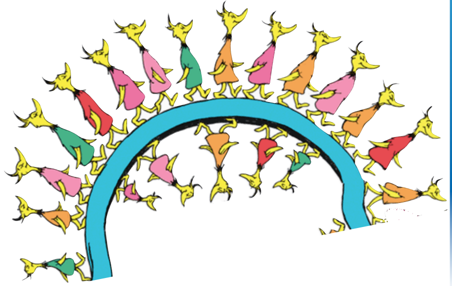
(Recall: contention = sum of broadcast probabilities.)



Batches based upon contention

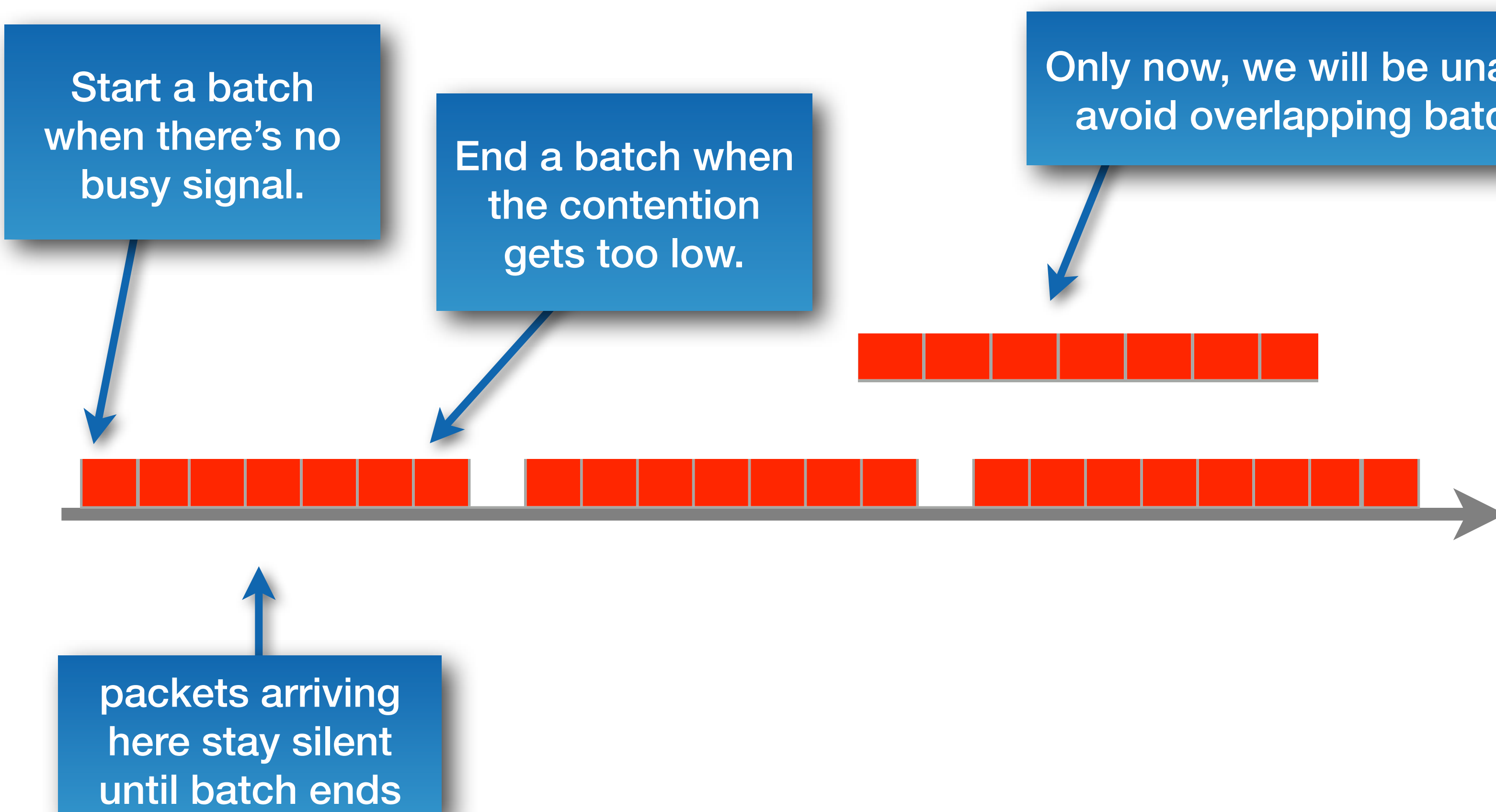
Group packets into synchronized batches.

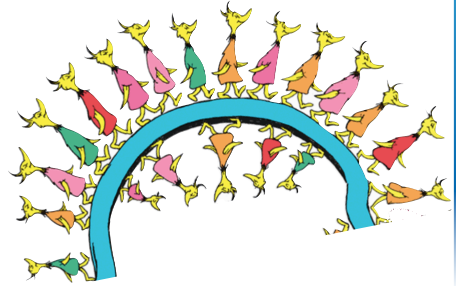




Batches based upon contention

Group packets into synchronized batches.





Managing Contention depends on age structure of packets

How contention changes depends on the age structure of the packets.

young packets:

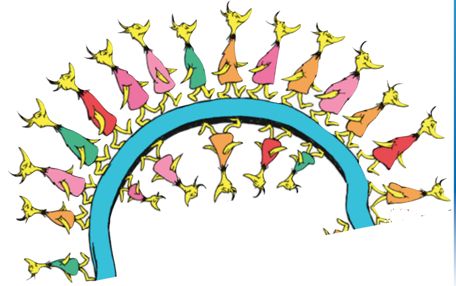
- create a lot of contention,
- but their contention reduces *quickly* as they age.

1 → 1/2 → 1/3 → 1/4 → 1/5 ...

old packets:

- create little contention,
- but their contention reduces *slowly* as they age.

1/1000 → 1/1001 → 1/1002 → 1/1003 → 1/1004 ...



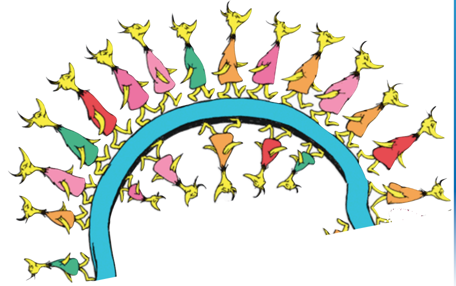
Resolving TDB

For a request that has been *active* for s slots:

- **Broadcast on the control channel with prob $\Theta(\log s / s)$.**
- **Broadcast on the data channel with prob $\Theta(1 / s)$:**
- **If successful, terminate.**
- **If 7/8ths of the s slots are empty, then become *inactive*.**

For an *inactive* request:

- **Wait until the first "silent" slot on the control channel.**
- **Become *active*.**



Resolving TDB

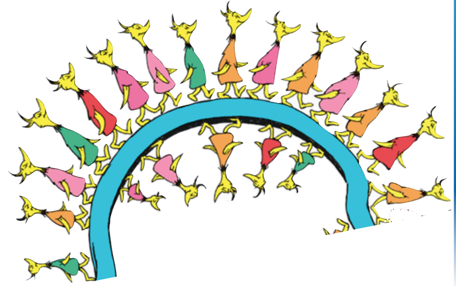
Cheap probabilistic busy signal.

For a request that has been *active* for s slots:

- **Broadcast on the control channel with prob $\Theta(\log s / s)$.**
- **Broadcast on the data channel with prob $\Theta(1 / s)$:**
- **If successful, terminate.**
- **If 7/8ths of the s slots are empty, then become *inactive*.**

For an *inactive* request:

- **Wait until the first “silent” slot on the control channel.**
- **Become *active*.**



Resolving TDB

For a request that has been *active* for *s* slots:

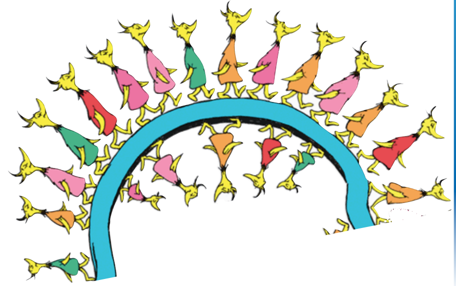
- **Broadcast on the control channel with prob $\Theta(\log s / s)$.**
- **Broadcast on the data channel with prob $\Theta(1 / s)$:**
- **If successful, terminate.**
- **If 7/8ths of the *s* slots are empty, then become *inactive*.**

Cheap probabilistic busy signal.

Just like exponential backoff.

For an *inactive* request:

- **Wait until the first "silent" slot on the control channel.**
- **Become *active*.**



Resolving TDB

For a request that has been *active* for *s* slots:

- Broadcast on the control channel with prob $\Theta(\log s / s)$.
- Broadcast on the data channel with prob $\Theta(1 / s)$:
- If successful, terminate.
- If 7/8ths of the *s* slots are empty, then become *inactive*.

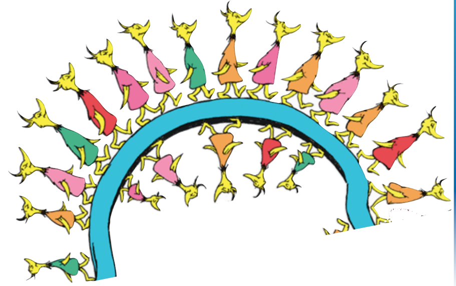
Cheap probabilistic busy signal.

Just like exponential backoff.

Fault-tolerant measure of low contention.
A batch ends when O(1) fraction of packets finished.

For an *inactive* request:

- Wait until the first "silent" slot on the control channel.
- Become *active*.



Resolving TDB

For a request that has been *active* for *s* slots:

- <u>Broadcast</u> on the <u>control channel</u> with prob $\Theta(\log s / s)$.
- <u>Broadcast</u> on the <u>data channel</u> with prob $\Theta(1 / s)$:
- If successful, terminate.
- If 7/8ths of the *s* slots are empty, then become *inactive*.

Cheap probabilistic busy signal.

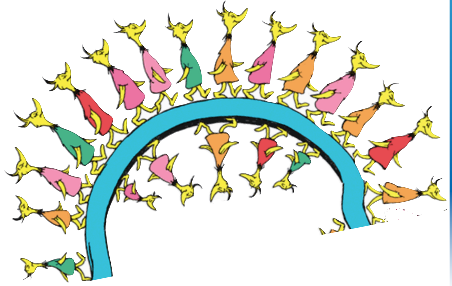
Just like exponential backoff.

Fault-tolerant measure of low contention.
A batch ends when O(1) fraction of packets finished.

For an *inactive* request:

- Wait until the first "silent" slot on the control channel.
- Become *active*.

Start a new batch.
(There may still be older batches in the system.)



What makes this analysis ~~irritating~~ ~~fun~~ ~~irritating~~ fun

Batches now overlap.

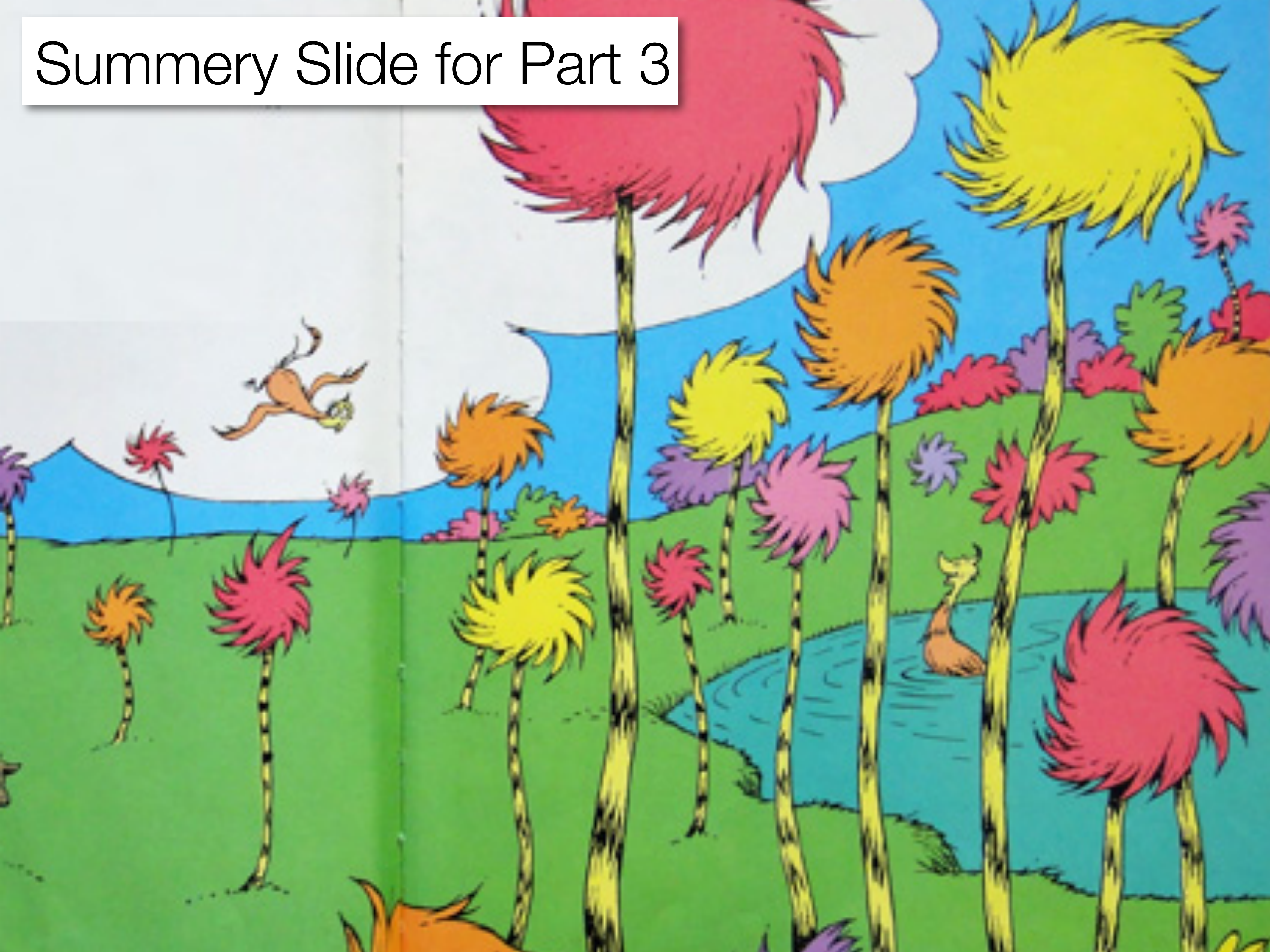
- Many batches are running simultaneously with different start times.

We can't use w.h.p. analysis on each batch.

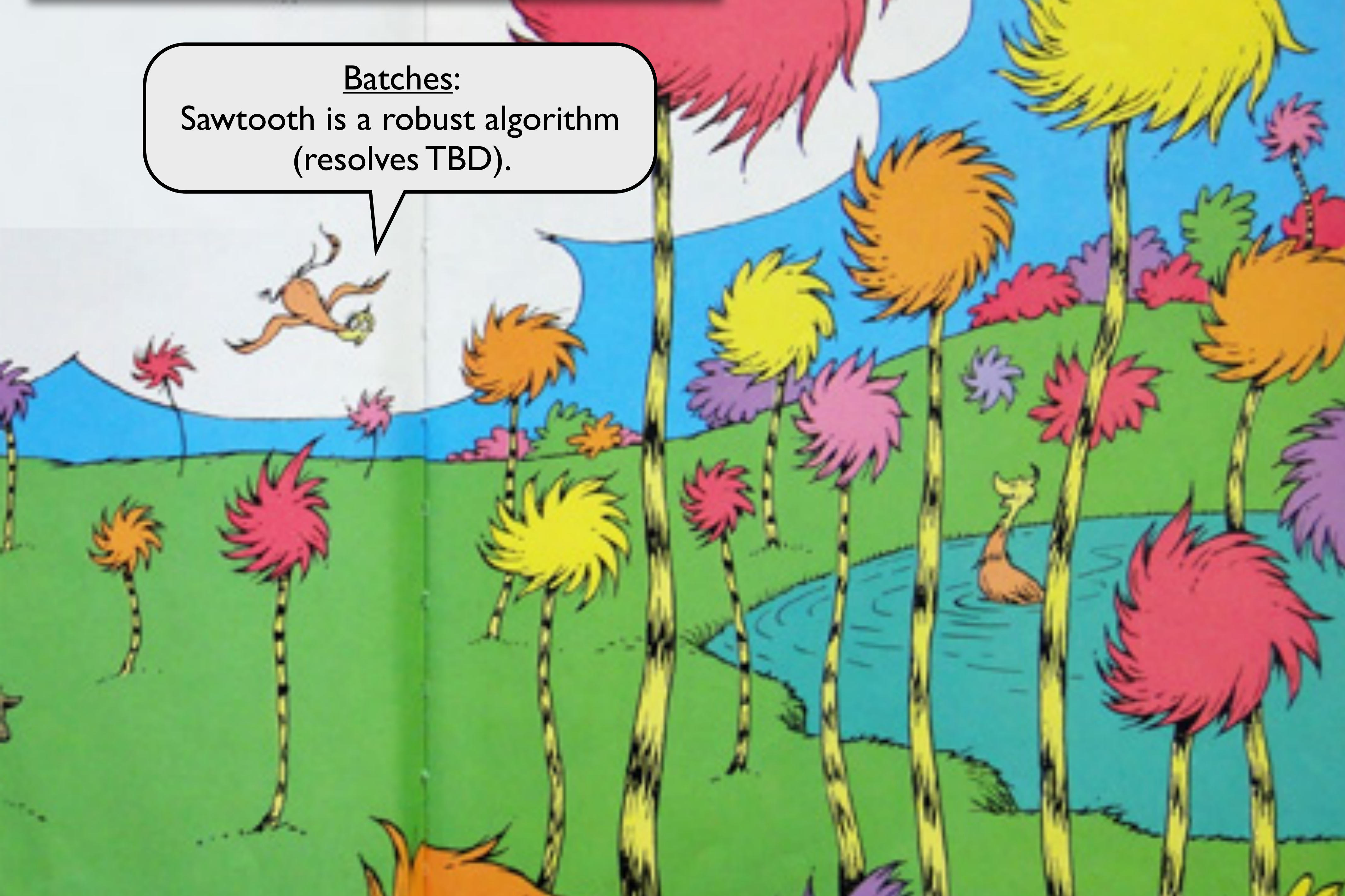
Contention is a slippery parameter.

- How contention changes depends on the age structure of the packet.

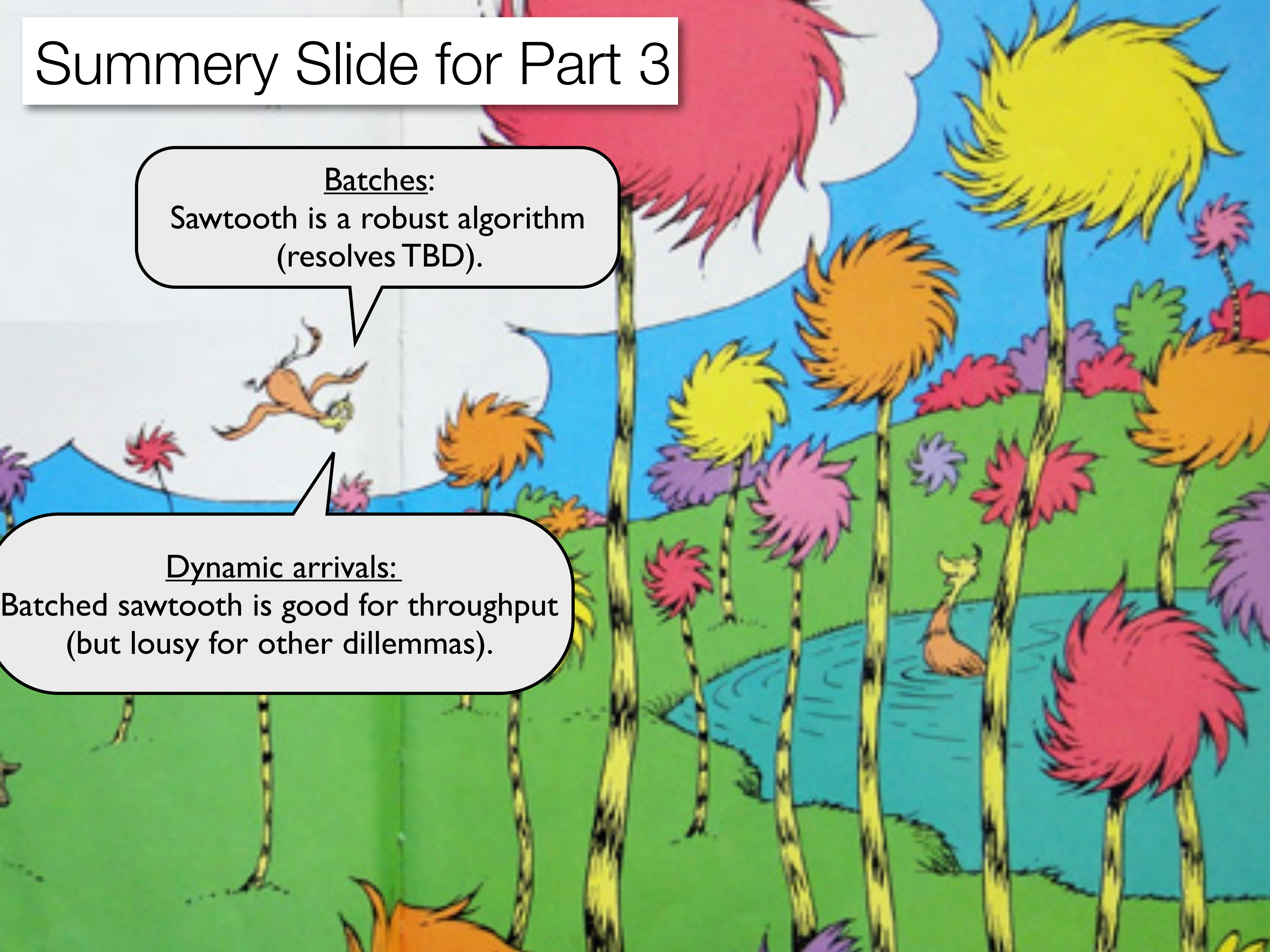
Summery Slide for Part 3



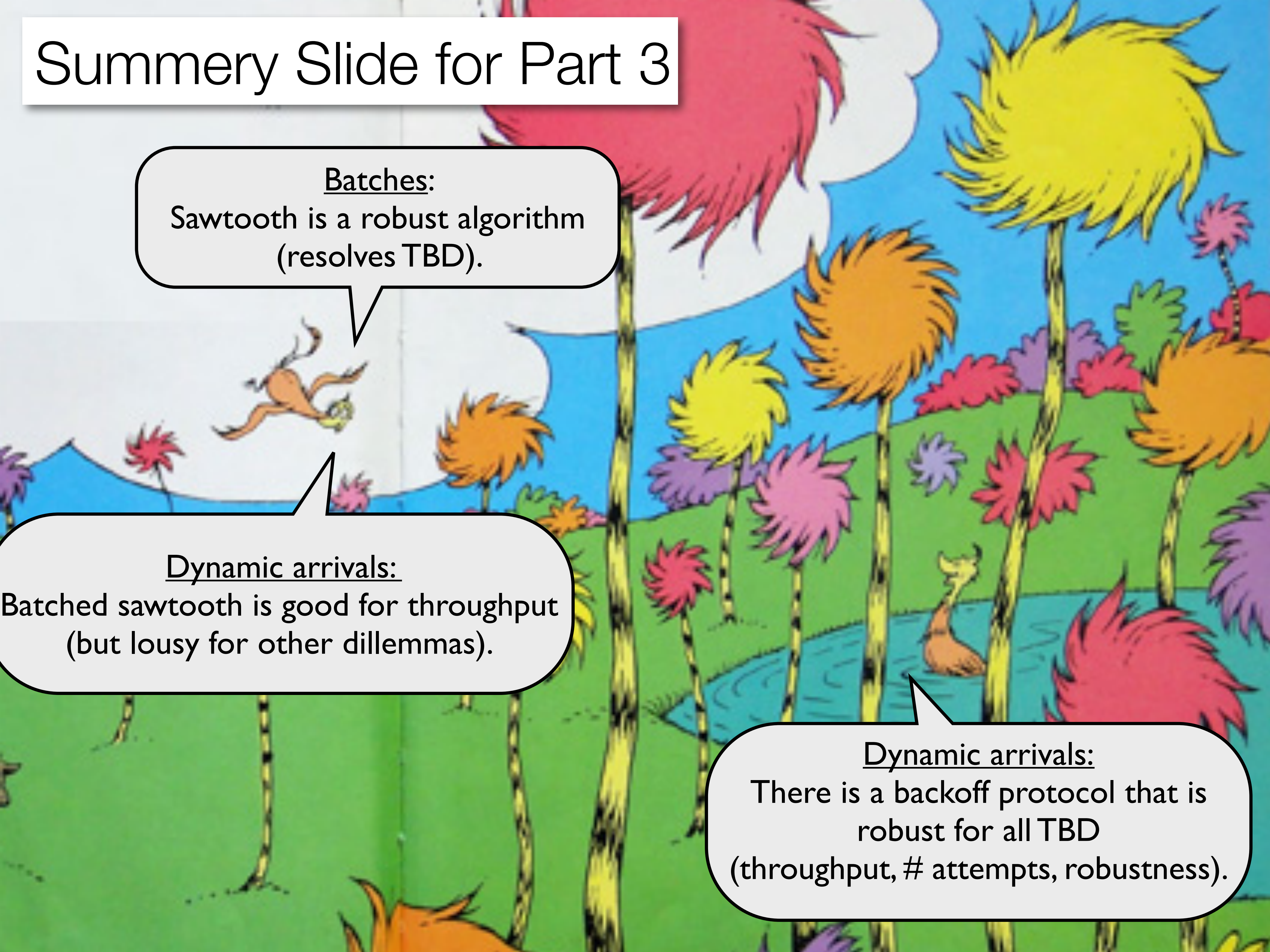
Summery Slide for Part 3



Summery Slide for Part 3



Summery Slide for Part 3



Batches:
Sawtooth is a robust algorithm (resolves TBD).

Dynamic arrivals:
Batched sawtooth is good for throughput (but lousy for other dillemmas).

Dynamic arrivals:
There is a backoff protocol that is robust for all TBD (throughput, # attempts, robustness).

Strive for backoff protocols that scale

Exponential backoff is broken (but ubiquitous)

- batch--backs off too quickly
- dynamic arrivals--doesn't deal well with bursts.

TBD

- minimize throughput
- maximize # attempts to access channel
- achieve robustness

Fixing exp backoff

- batch--sawtooth backoff resolves the TBDs.
- dynamic arrivals--sawtooth + busy tone has good throughput
- dynamic arrivals--cheap busytone + exponential backoff + delayed reset + lots of analysis resolves the TBDs.
  - Not complicated algorithm. complicated analysis.