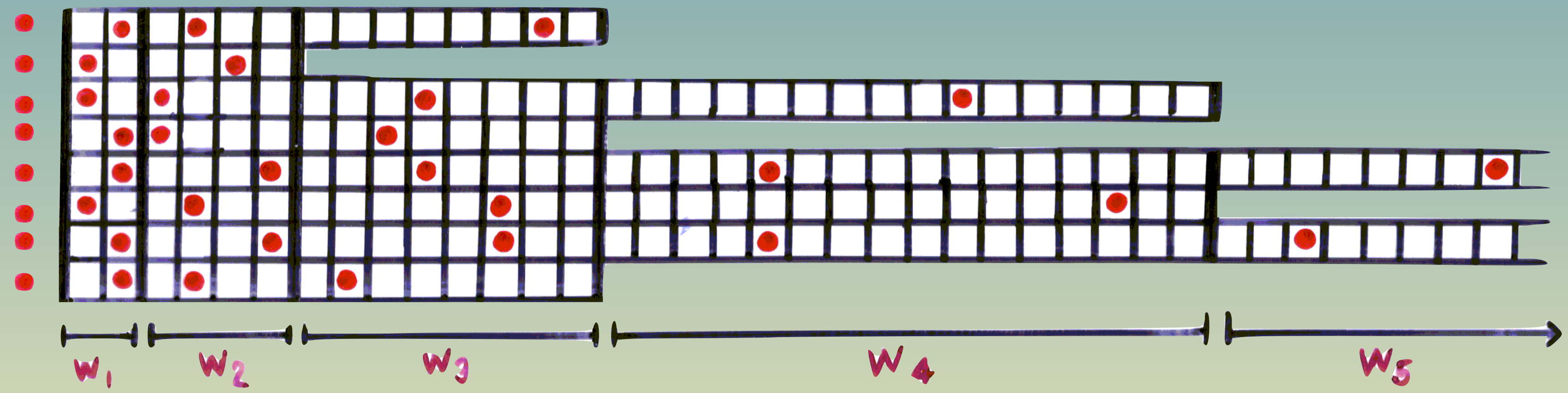
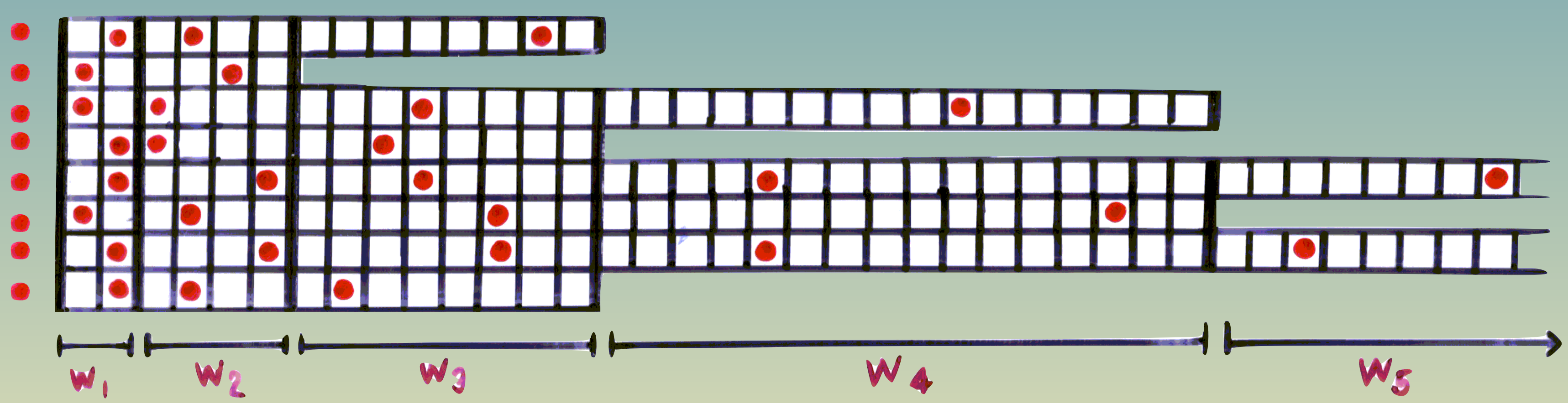


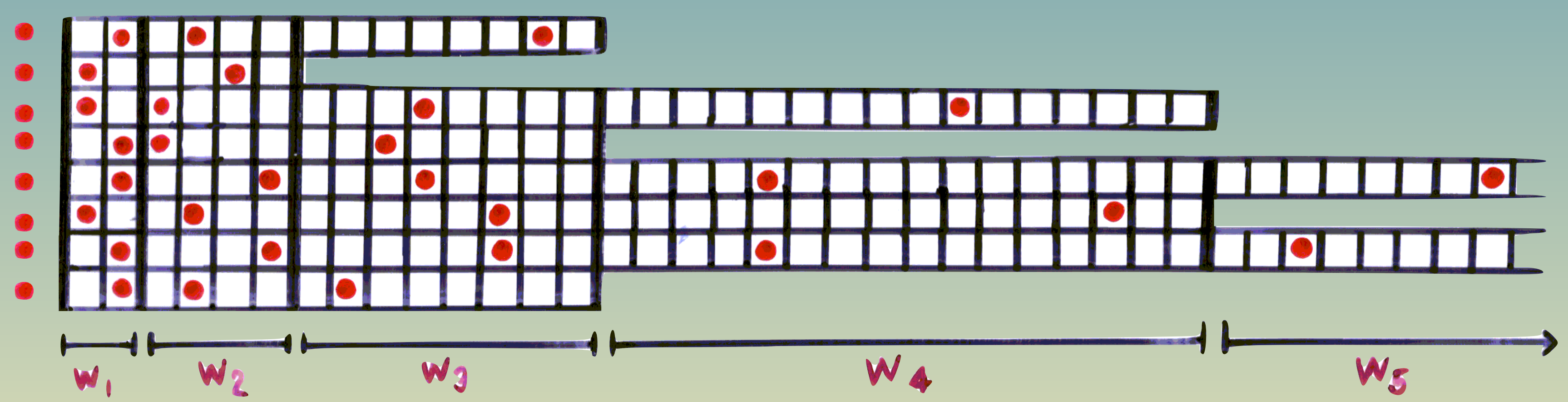
w_1
w_2
w_3
w_4
w_5

TBD



TBD
Three backoff dilemmas

Michael A. Bender

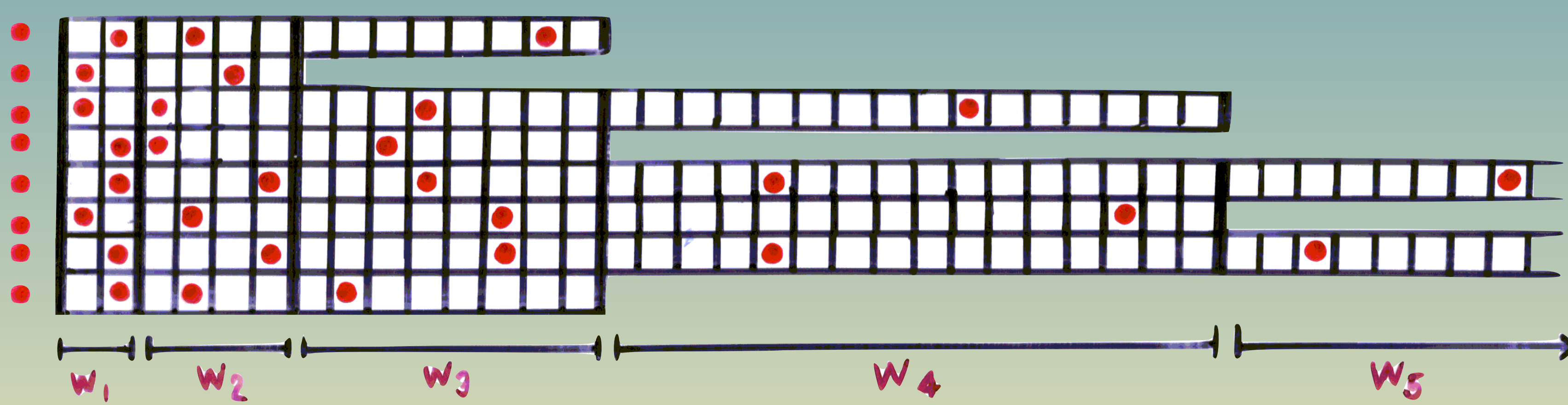


TBD

Three backoff dilemmas

Michael A. Bender

Joint work with Jeremy Fineman, Seth Gilbert, Tsvi Kopelowitz, Seth Pettie, and Maxwell Young.



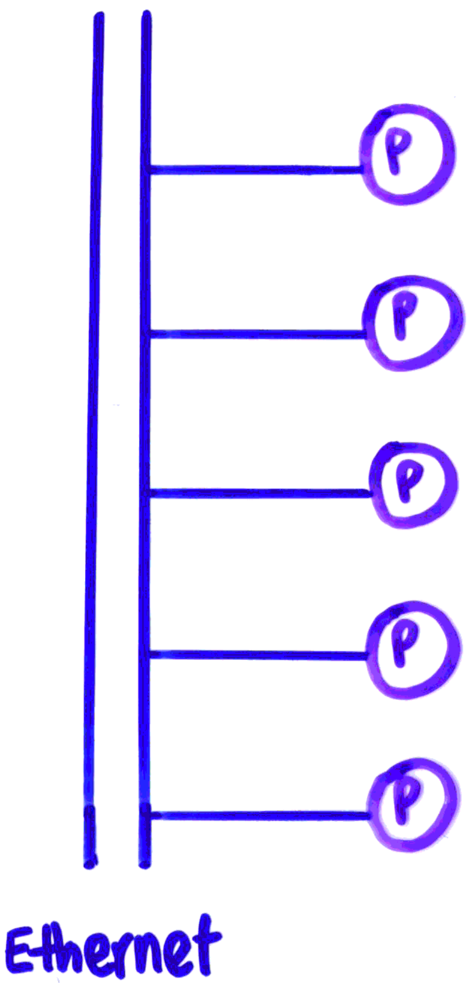
Backoff is about sharing

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)



Backoff as scheduling problem

packets

- unit length jobs

shared channel

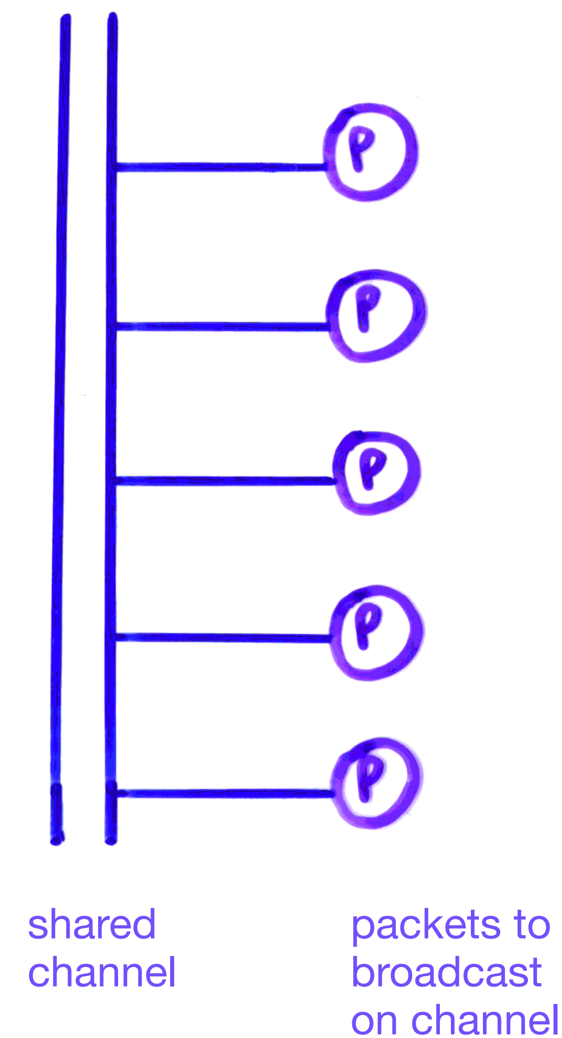
- single "processor"

objective: minimize makespan

- broadcast all packets on channel to maximize throughput

scheduling subtlety: backoff mechanism

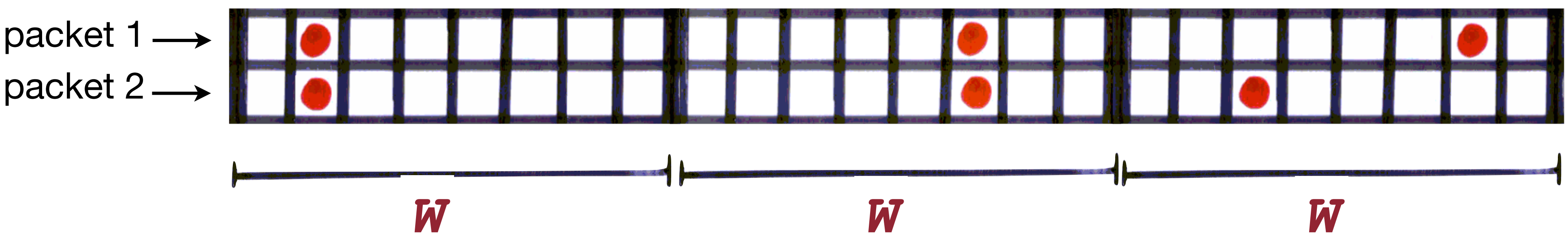
- how to coordinate access to channel



Randomized backoff [Abramson '70]

Repeat until successful transmission

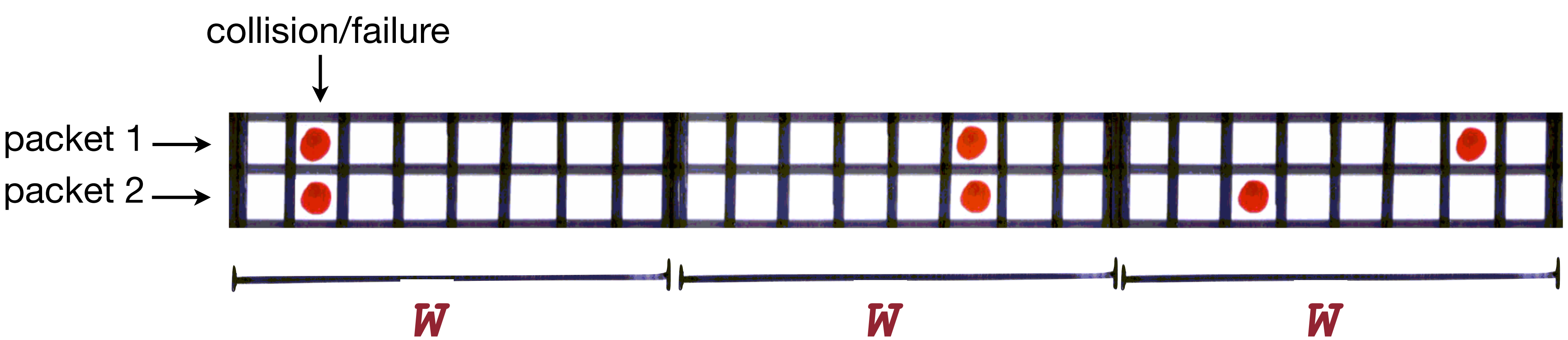
- **Try to broadcast**
- **If failure then**
  randomly choose *t* in window *W*
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until successful transmission

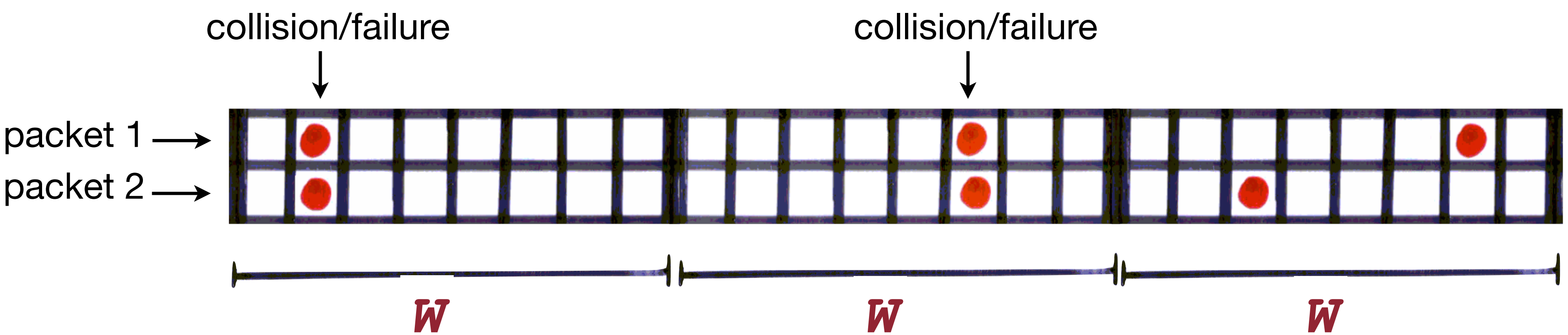
- **Try to broadcast**
- **If failure then**
  randomly choose *t* in window *W*
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until successful transmission

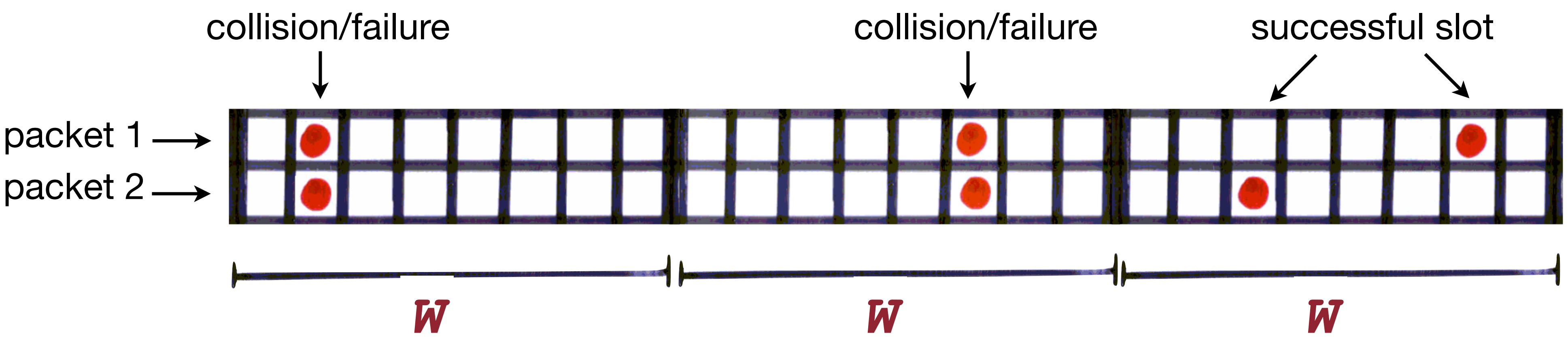
- **Try to broadcast**
- **If failure then**
  randomly choose *t* in window *W*
  and wait *t* seconds.



Randomized backoff [Abramson '70]

Repeat until successful transmission

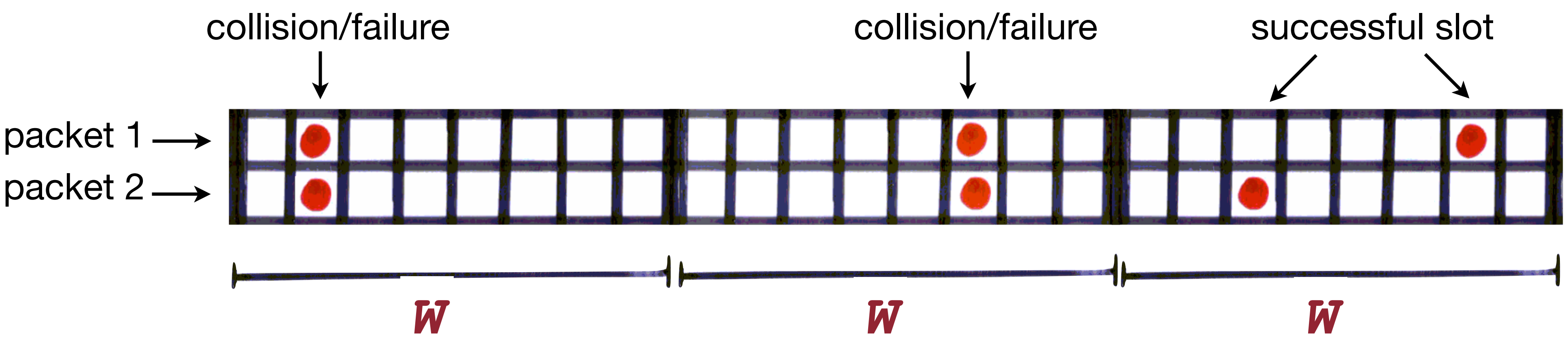
- **Try to broadcast**
- **If failure then**
  randomly choose *t* in window *W*
  and wait *t* seconds.



Randomized backoff [Abramson ’70]

Repeat until successful transmission

- **Try to broadcast**
- **If failure then**
  randomly choose *t* in window *W*
  and wait *t* seconds.

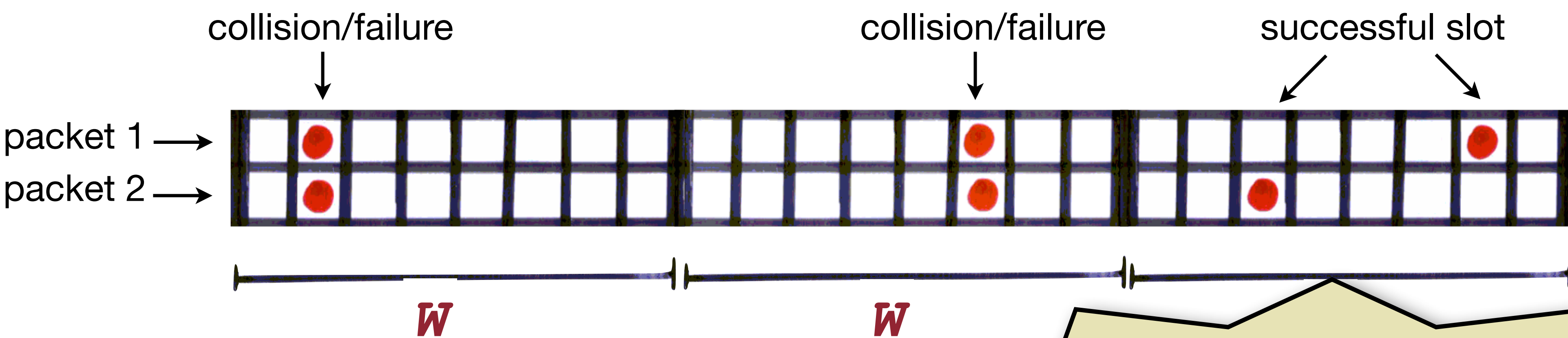


Bad scenario: thousands of devices contending for the channel.

Randomized backoff [Abramson '70]

Repeat until successful transmission

- **Try to broadcast**
- **If failure then**
  randomly choose t in window W
  and wait t seconds.



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

Bad scenario: thousands of devices contending for the channel.

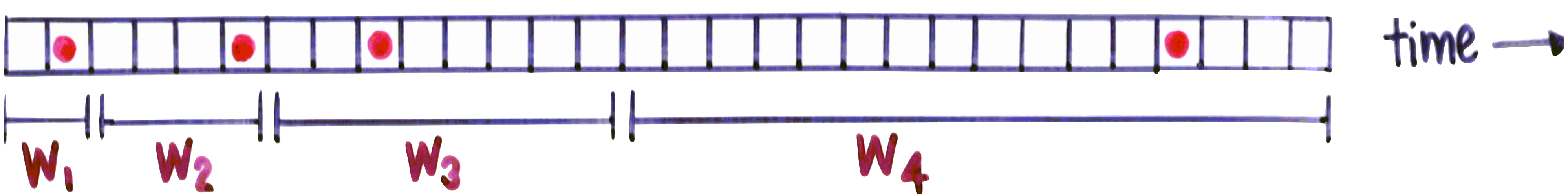
Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

Window size $W = 2$

Repeat until successful transmission:

- Randomly choose slot t in window.
- Try to transmit at slot t.
- If failure, wait to end of W.
  Then double W.



Standard answer: Binary exponential backoff

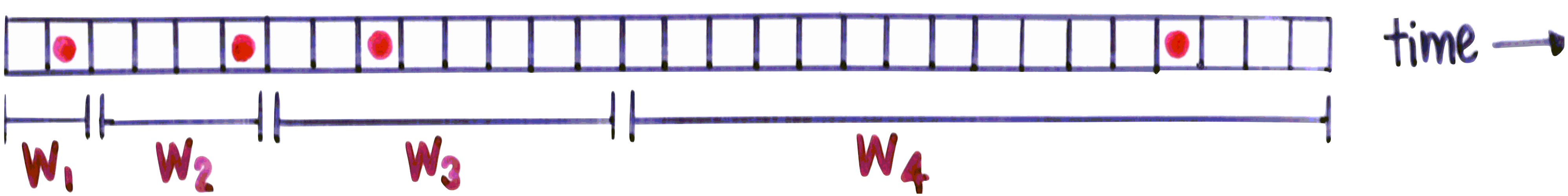
[Metcalfe and Boggs '76]

Window size $W = 2$

Repeat until successful transmission:

- **Randomly choose slot t in window.**
- **Try to transmit at slot t.**
- **If failure, 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]

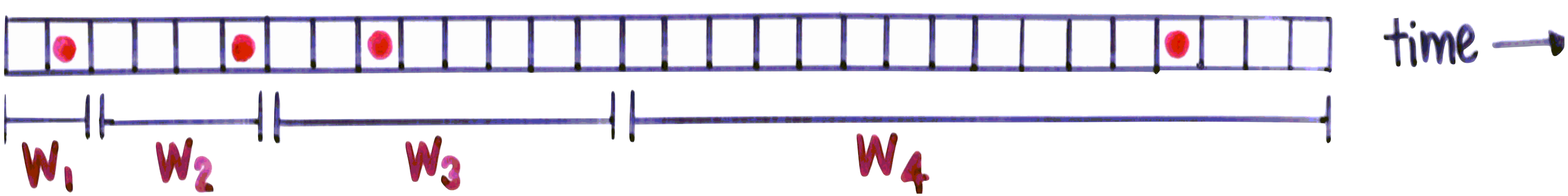
Window size *W* = 2

How many attempts until a success?

Repeat until successful transmission:

- Randomly choose slot *t* in window.
- Try to transmit at slot *t*.
- If failure, 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]

How many attempts until a success?

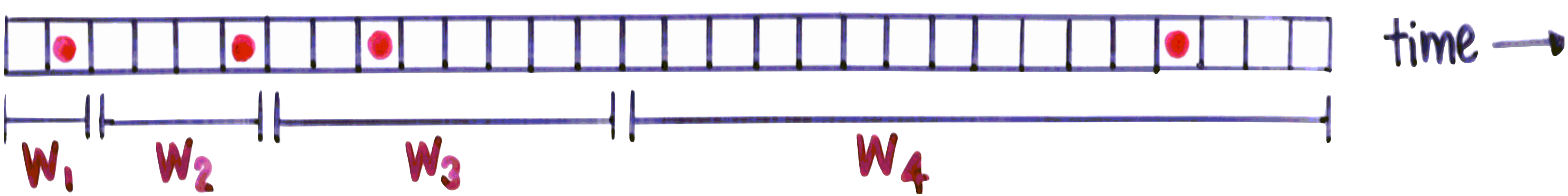
How well does exponential backoff deal with arbitrary release times?

Window size *W* = 2

Repeat until successful transmission:

- **Randomly choose slot *t* in window.**
- **Try to transmit at slot *t*.**
- **If failure, 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]

How many attempts until a success?

How well does exponential backoff deal with arbitrary release times?

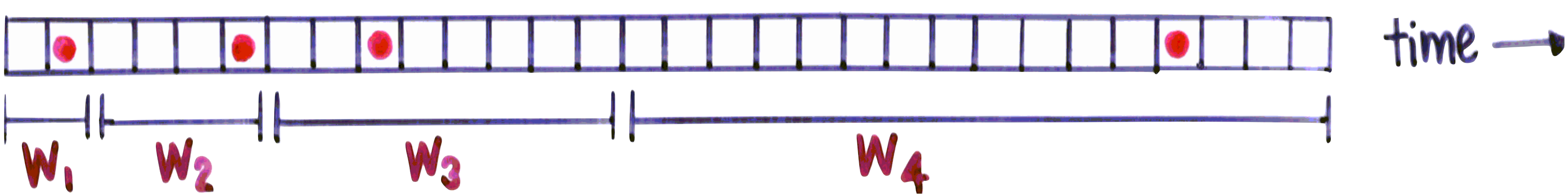
Window size $W = 2$

Repeat until successful transmission:

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

Are there any guarantees on makespan and throughput?

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



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

How many attempts until a success?

How well does exponential backoff deal with arbitrary release times?

Window size $W = 2$

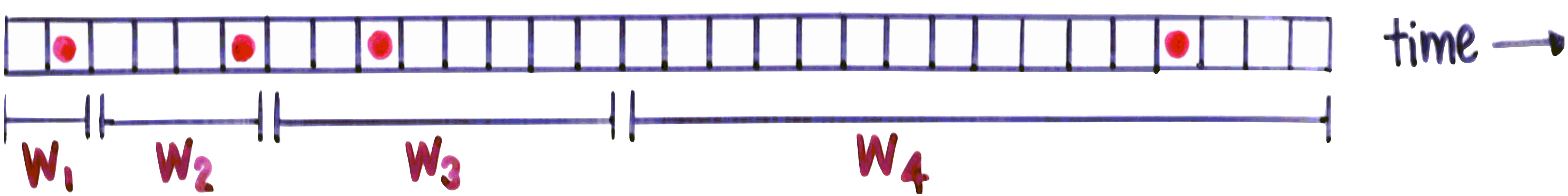
Repeat until successful transmission:

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

Are there any guarantees on makespan and throughput?

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

What about robustness guarantees?



Standard answer: Binary exponential backoff

[Metcalfe and Boggs '76]

How many attempts until a success?

How well does exponential backoff deal with arbitrary release times?

Window size *W* = 2

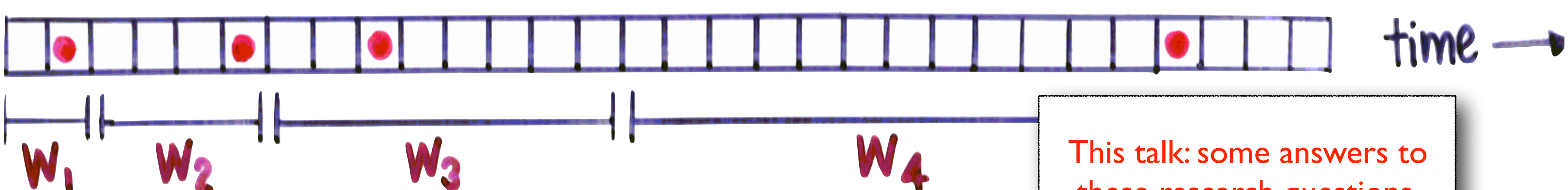
Repeat until successful transmission:

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

Are there any guarantees on makespan and throughput?

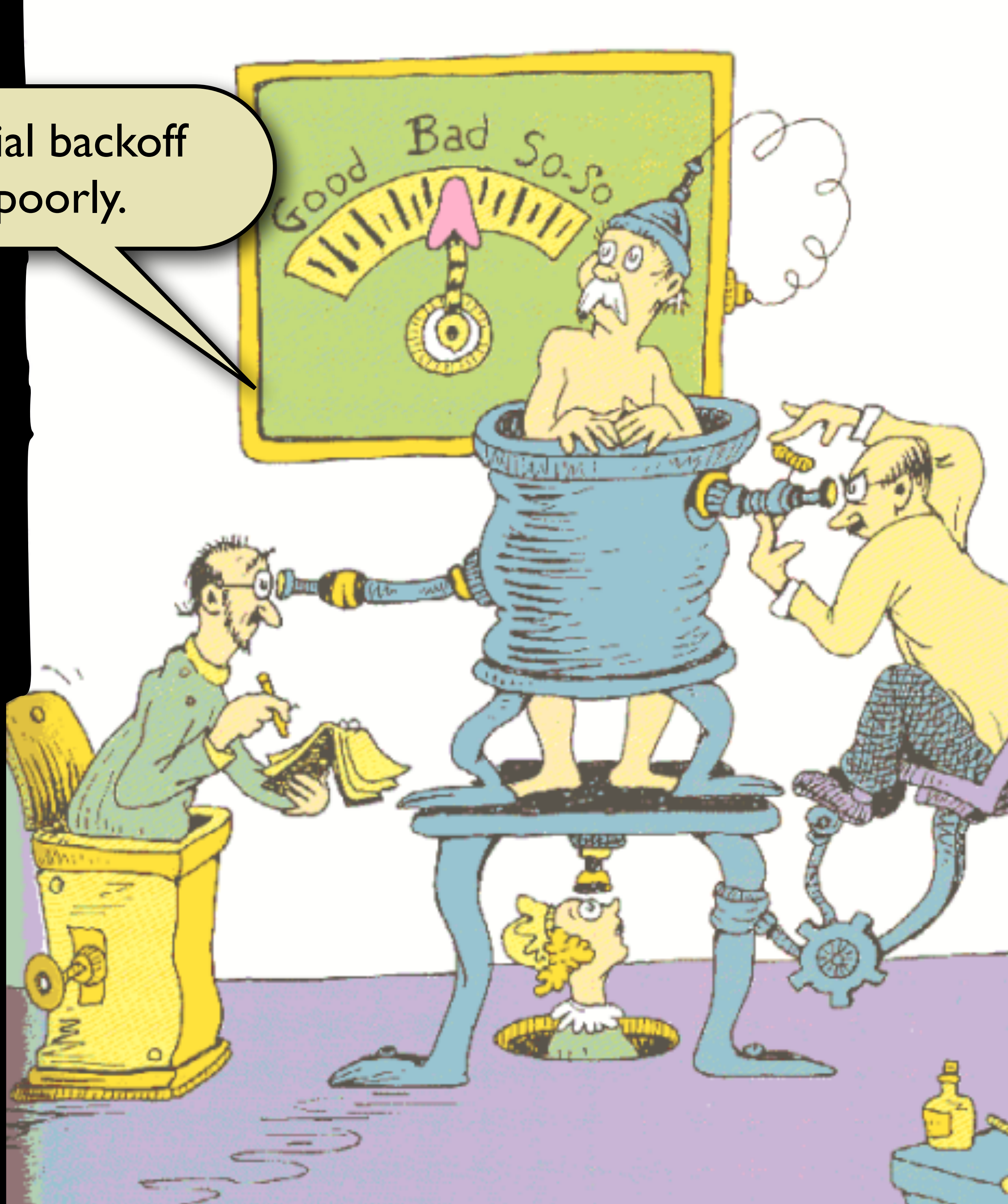
Why double?
What if the window size changes by a different factor?

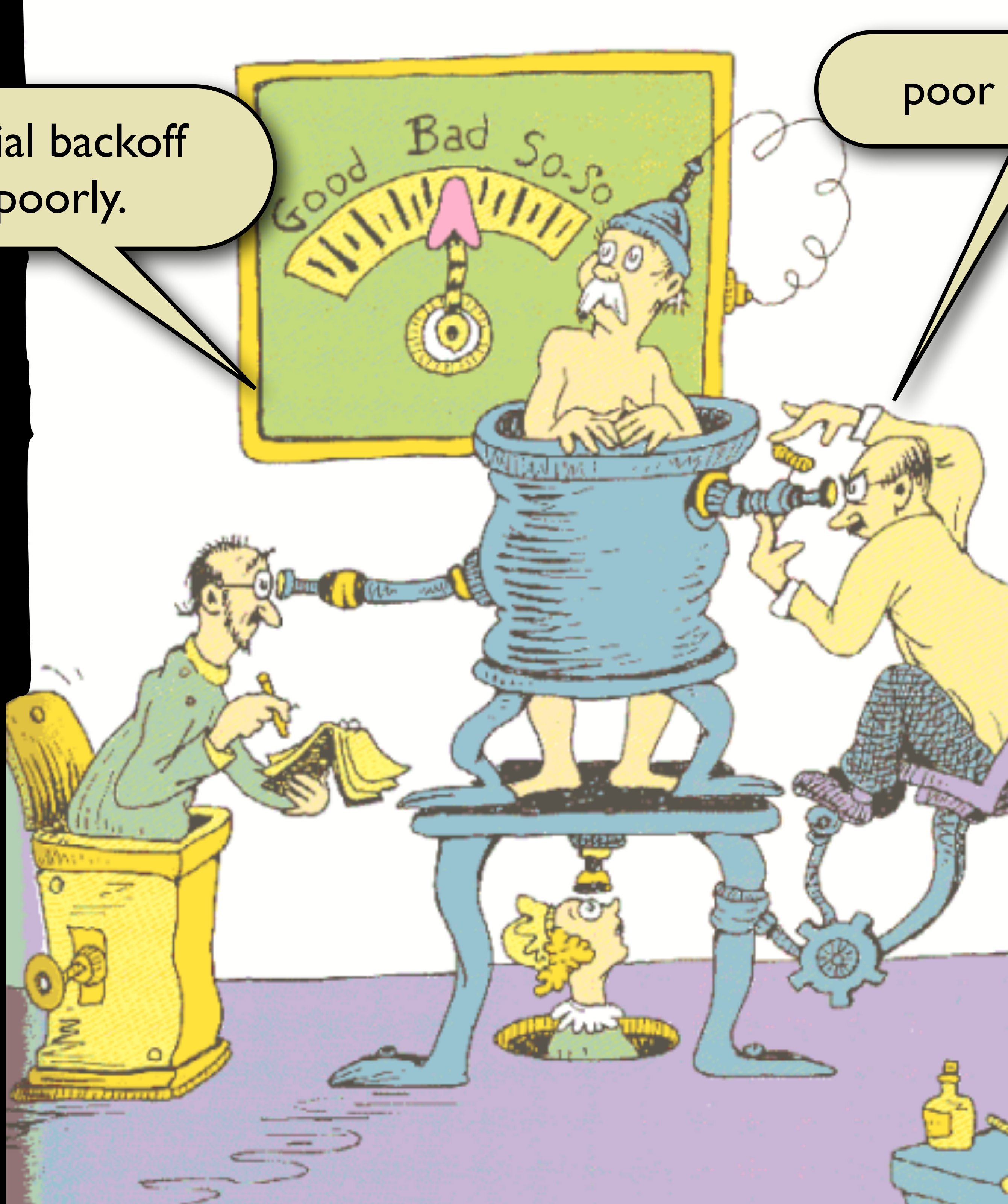
What about robustness guarantees?



This talk: some answers to these research questions.

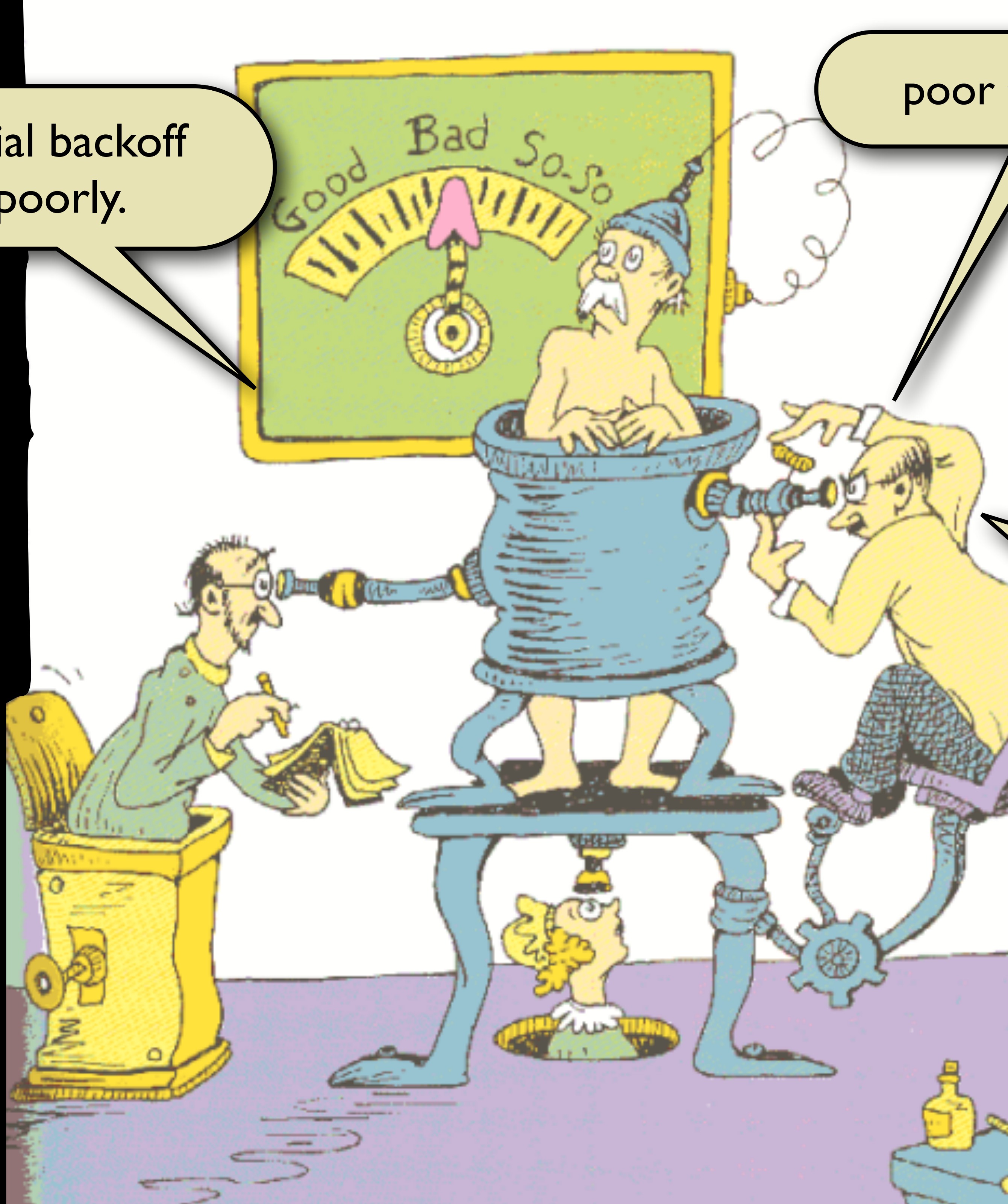
Exponential backoff scales poorly.



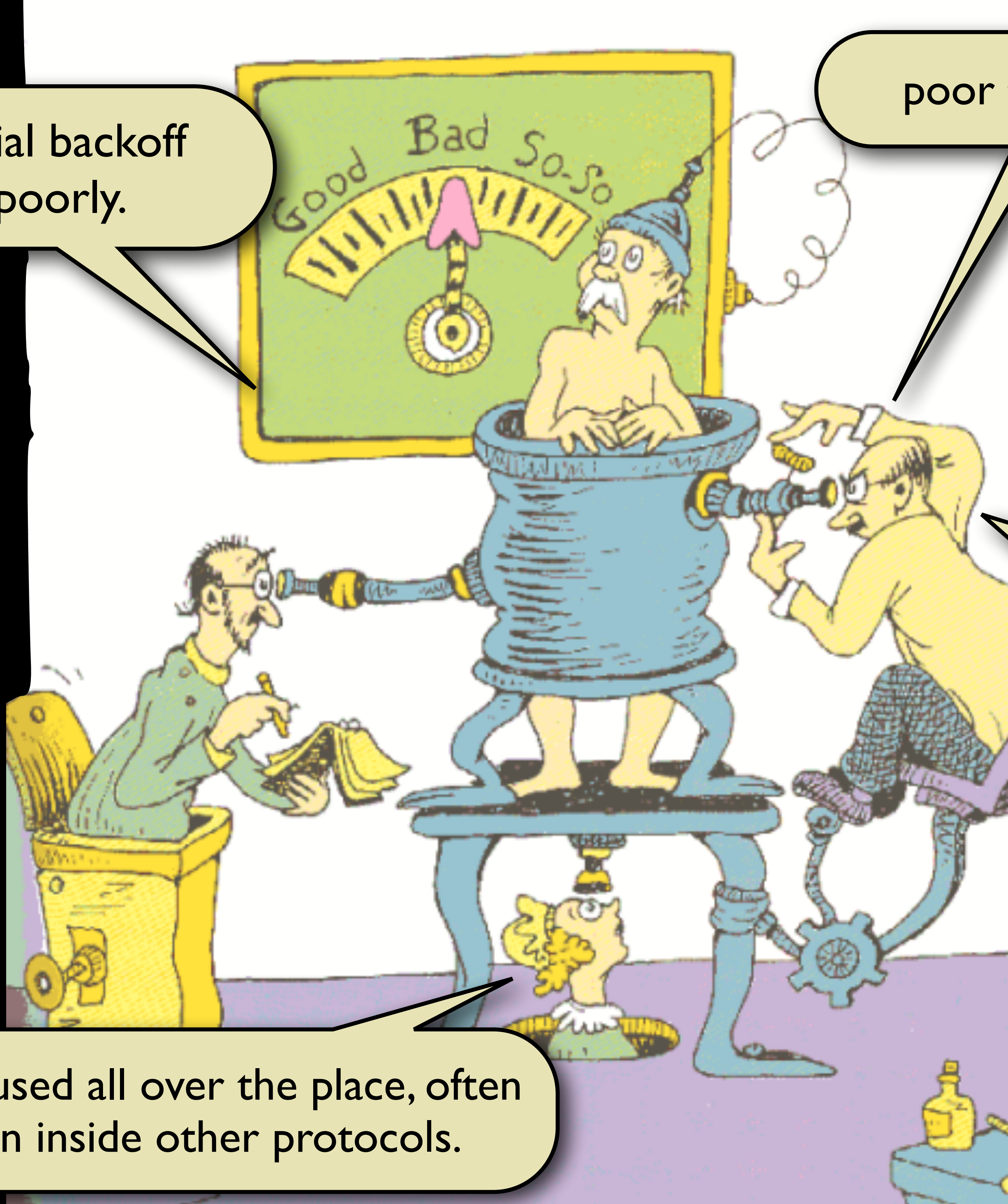

Exponential backoff scales poorly.
poor throughput
Good
Bad
So-So

Exponential backoff scales poorly.

poor throughput



fragile/not robust to failures

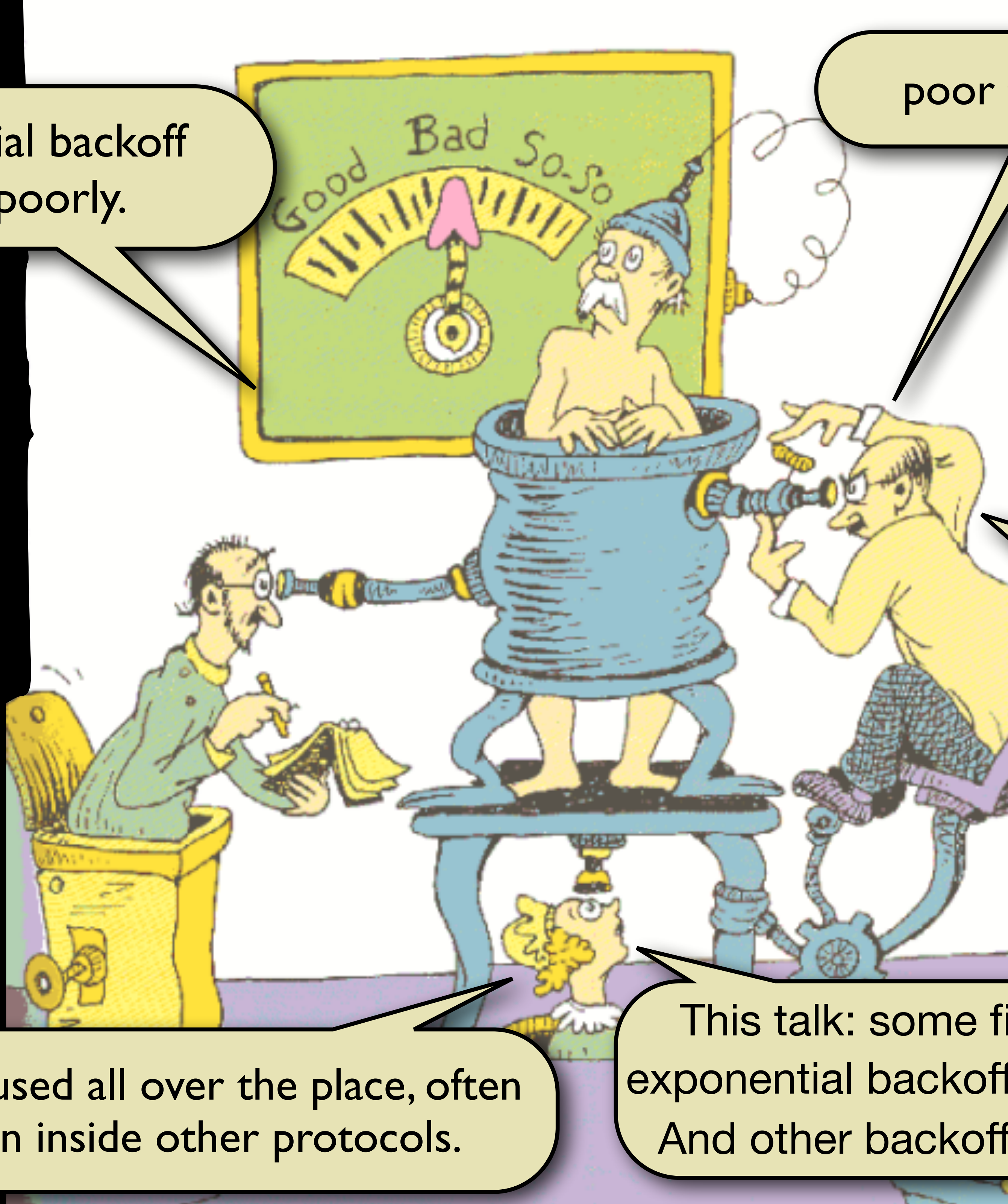


Exponential backoff scales poorly.

poor throughput

fragile/not robust to failures

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



Exponential backoff scales poorly.

poor throughput

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

TBD (three backoff dilemmas).

- minimize makespan (maximize throughput)
- minimize # tries to access resource (minimize energy)
- achieve robustness to jamming or failures

Binary exponential backoff scales poorly.

- batch (all release times = 0)
- dynamic arrivals (arbitrary release times)

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

Better randomized backoff algorithms

- batch
- dynamic arrivals

[Bender, Fineman, Gilbert, Young, SODA 16]

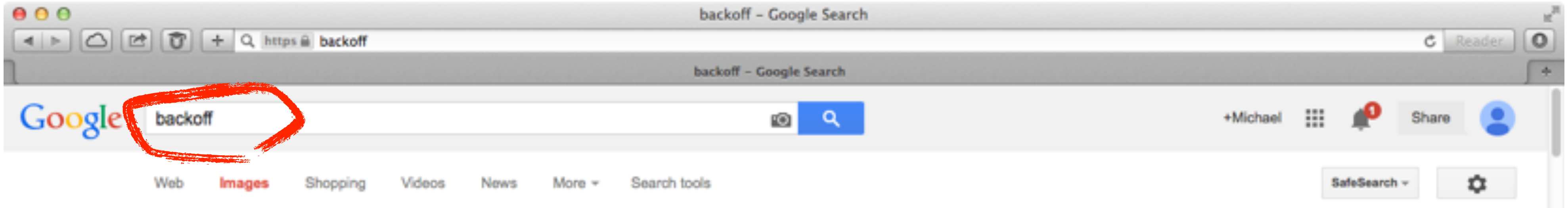
[Bender, Kopelowitz, Pettie Young STOC 16]

Good pictures help convey intuition.

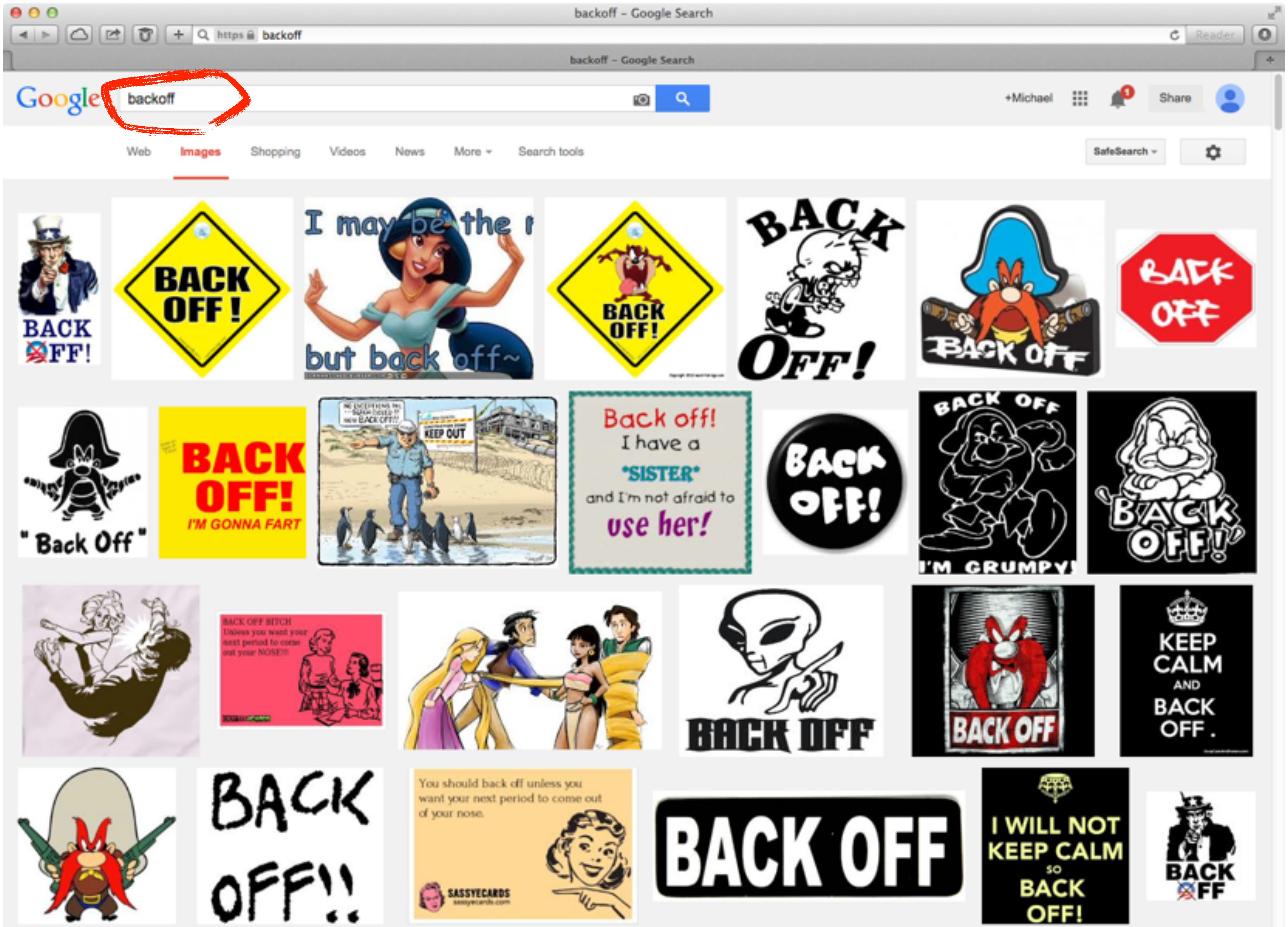
Good pictures help convey intuition.

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

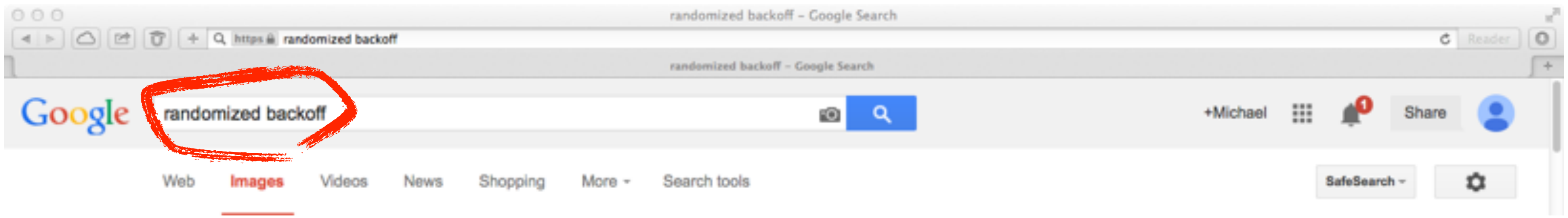
What Google says about backoff is intuitive but off topic.



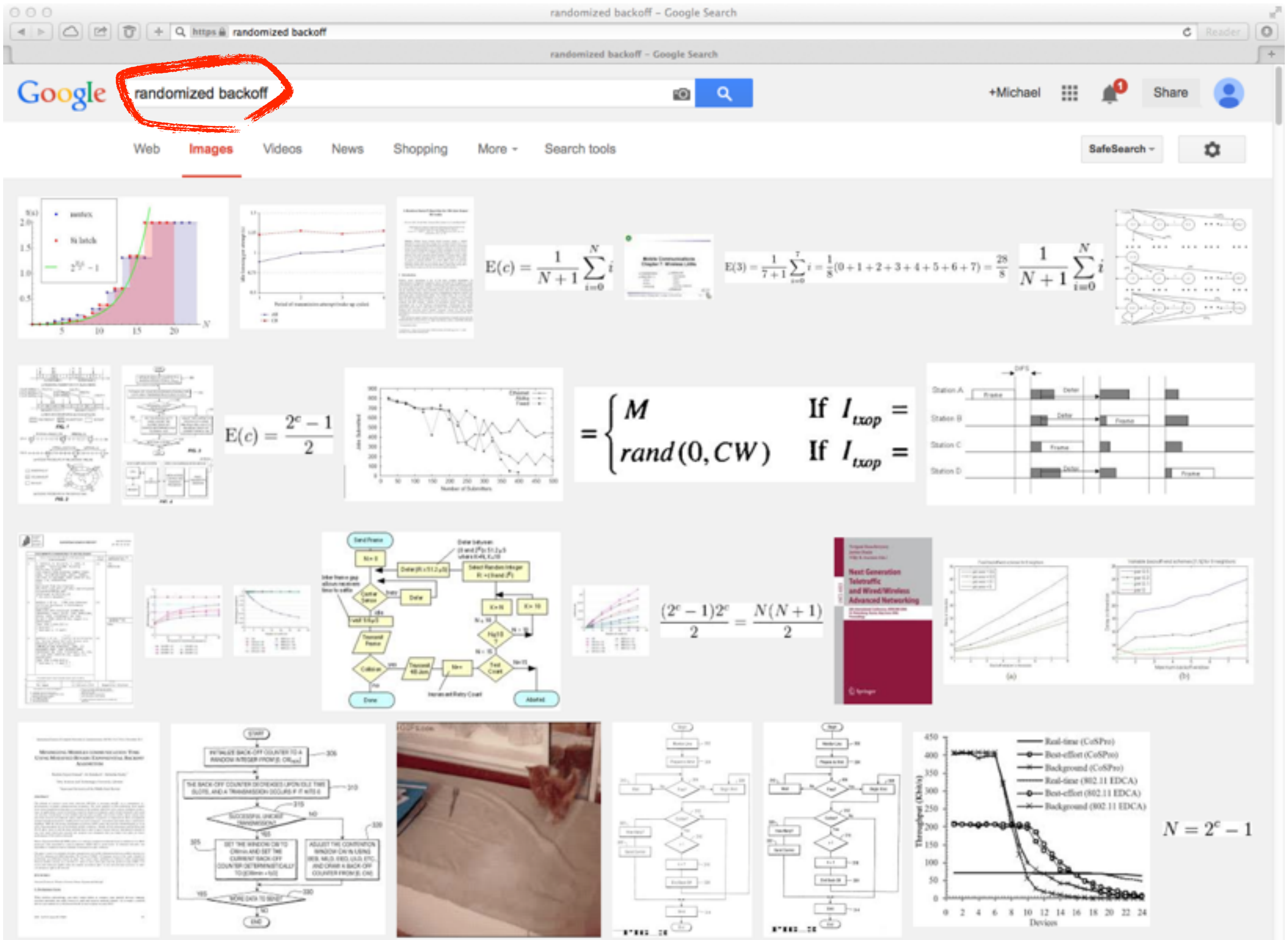
What Google says about backoff is intuitive but off topic.



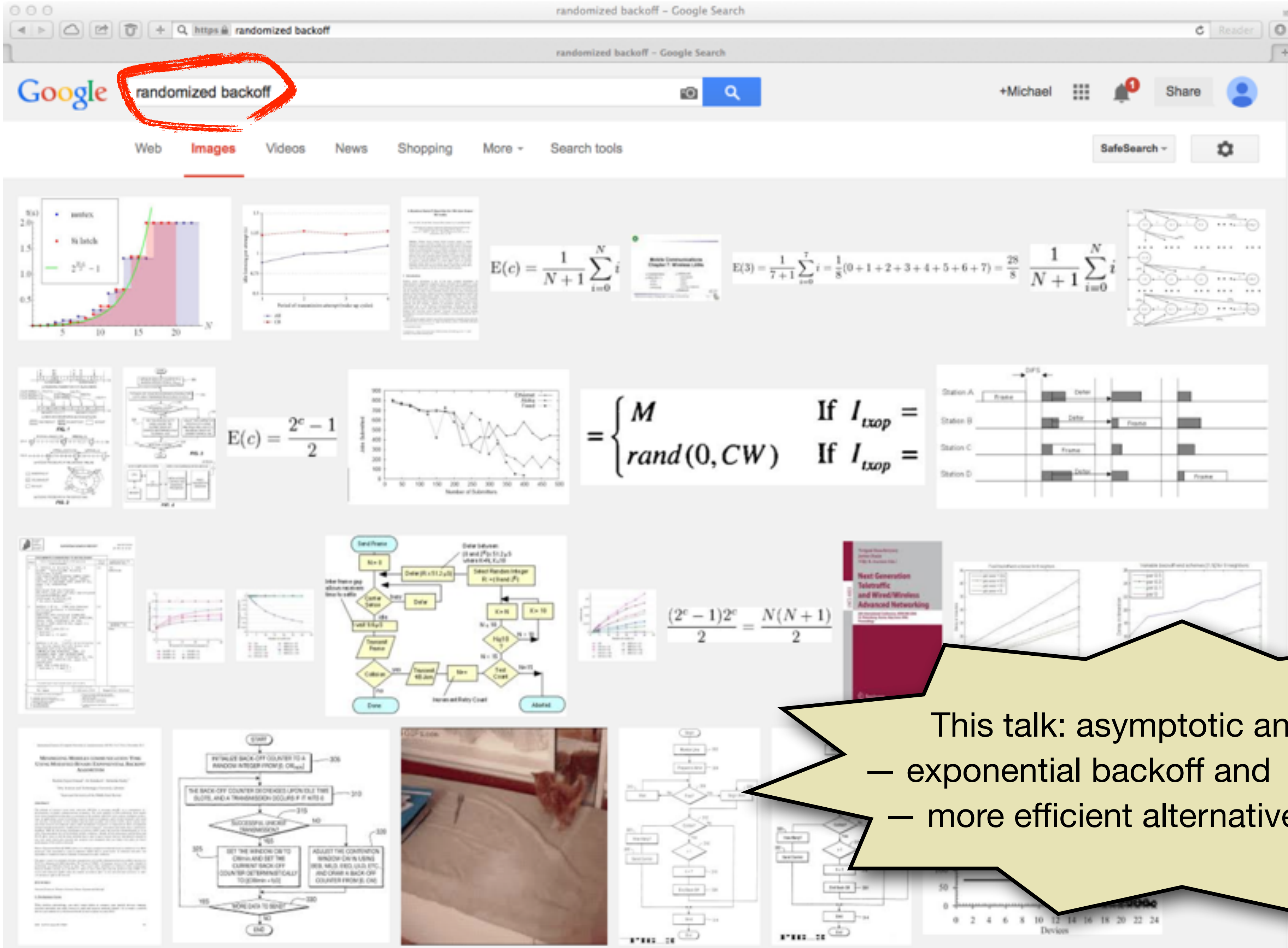
What Google says about "randomized backoff" is on topic but less algorithmic...



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



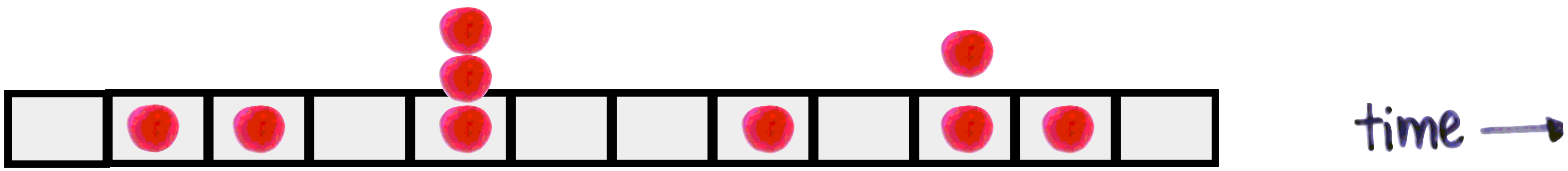
What Google says about "randomized backoff" is on topic but less algorithmic...



This talk: asymptotic analysis of
— exponential backoff and
— more efficient alternatives.

Model for multiple-access channels

Time is divided into discrete slots.

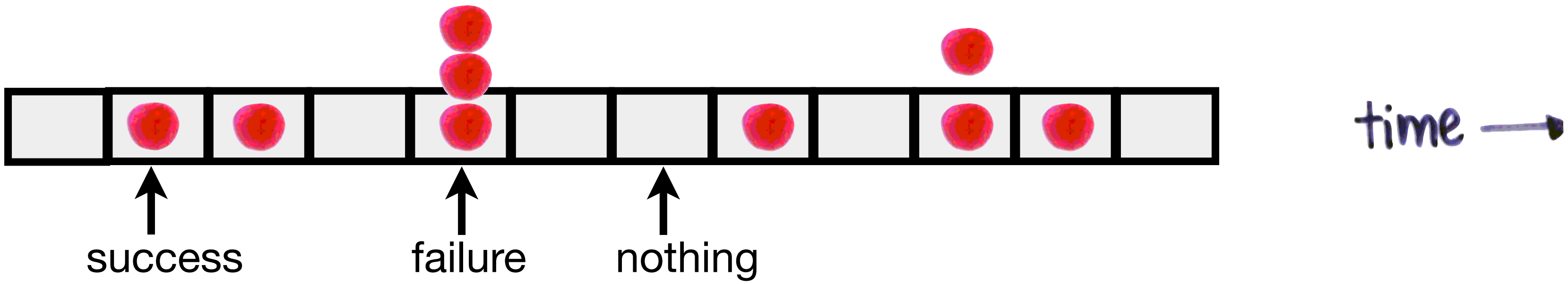


In every slot, a device can:

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

Model for multiple-access channels

Time is divided into discrete slots.



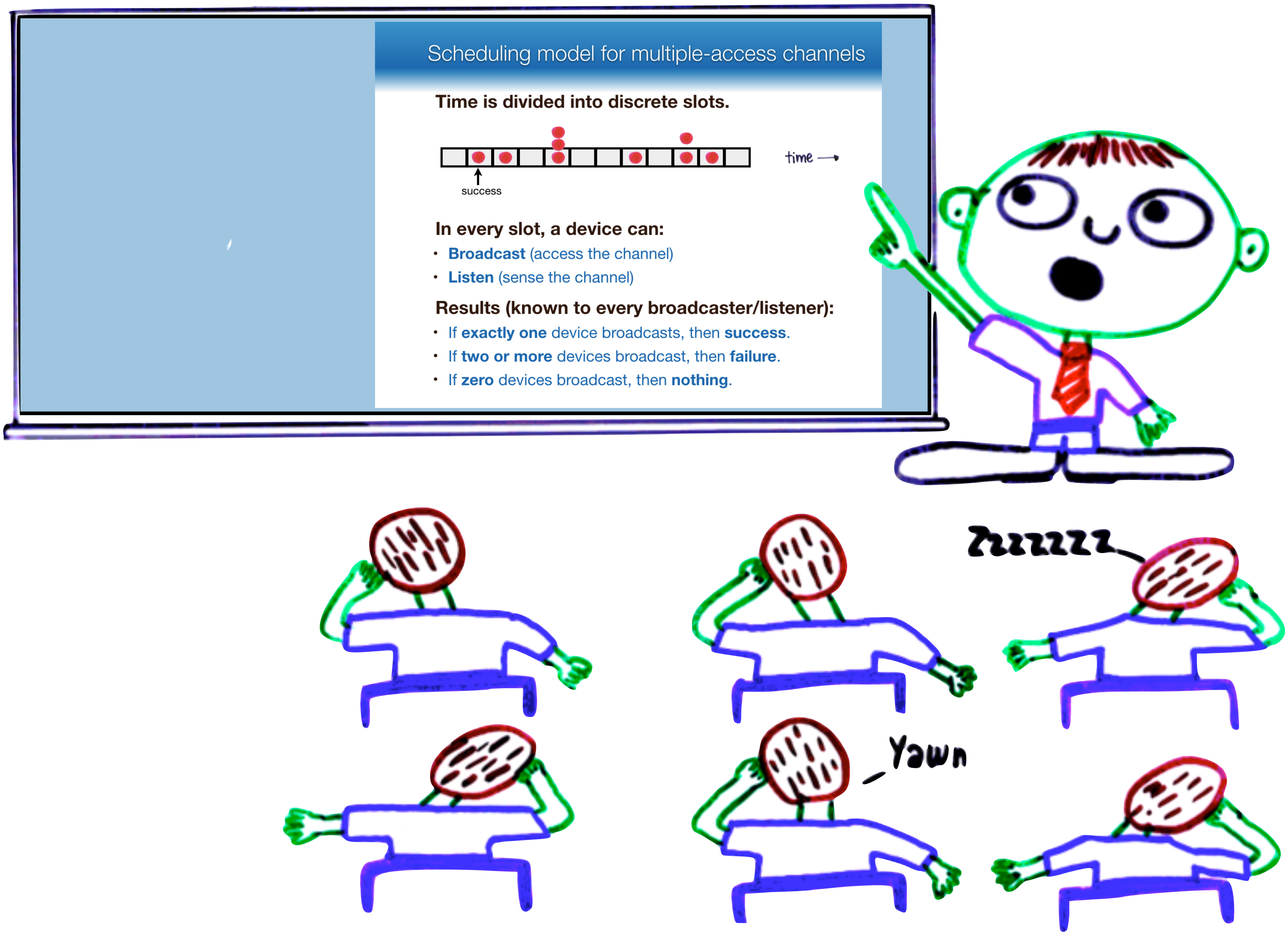
In every slot, a device can:

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

Results (known to every broadcaster/listener):

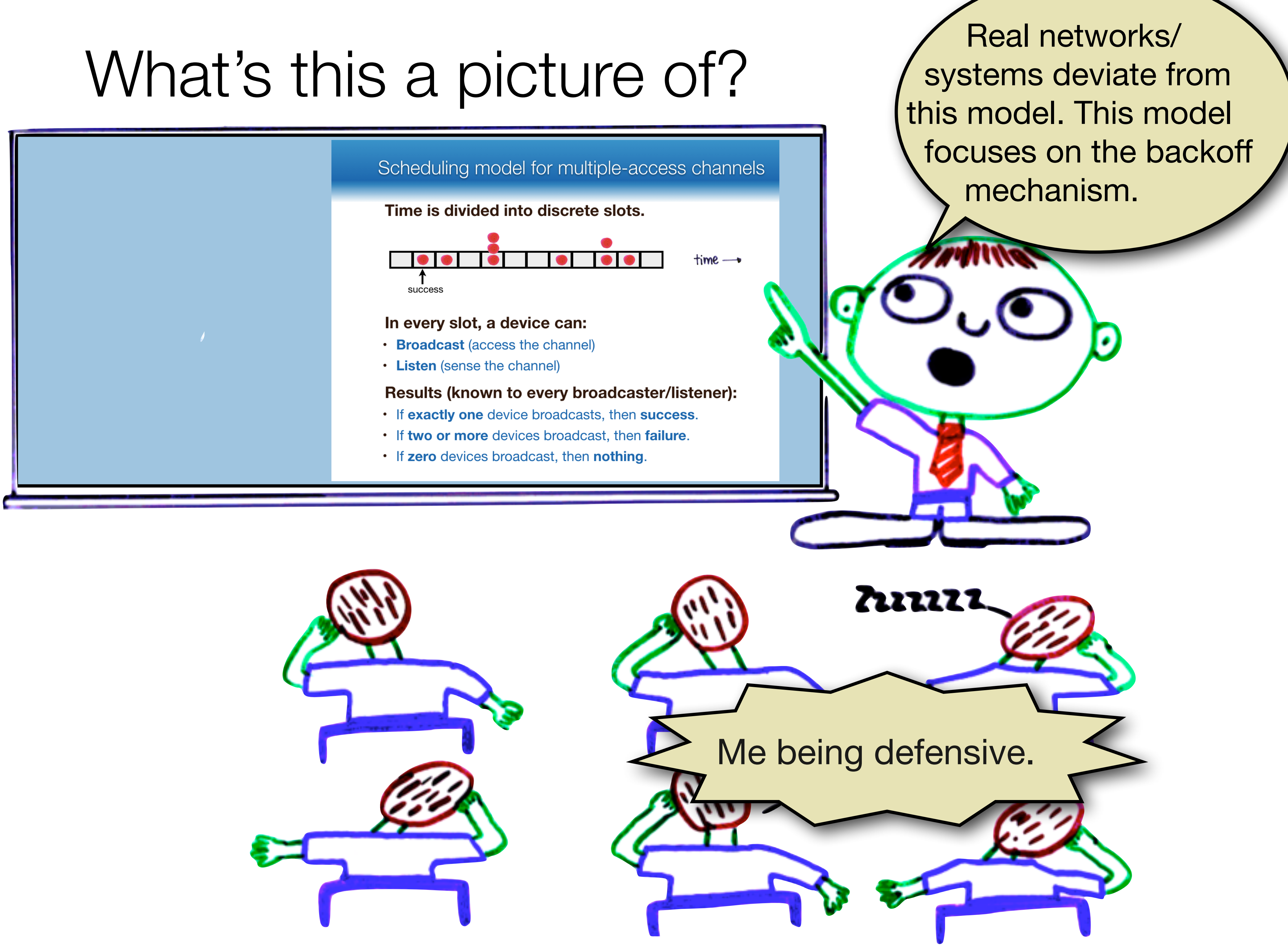
- If **exactly one** device broadcasts, then **success**.
- If **two or more** devices broadcast, then **failure**.
- If **zero** devices broadcast, then **nothing**.

What’s this a picture of?

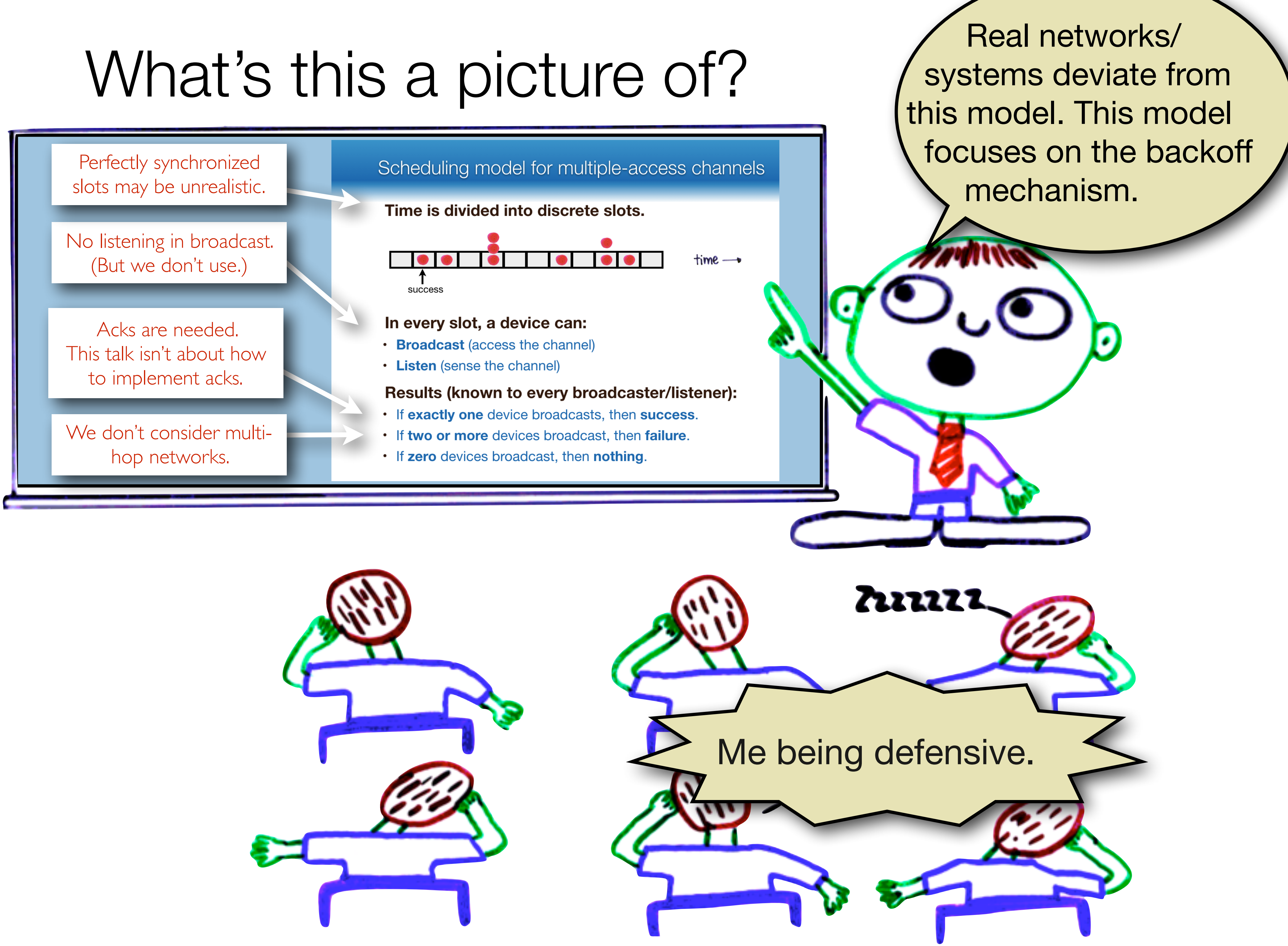


What’s this a picture of?

What’s this a picture of?



What’s this a picture of?



What's this a picture of?

Perfectly synchronized slots may be unrealistic.

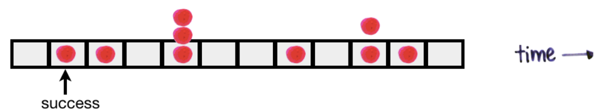
No listening in broadcast. (But we don't use.)

Acks are needed. This talk isn't about how to implement acks.

We don't consider multi-hop networks.

Scheduling model for multiple-access channels

Time is divided into discrete slots.

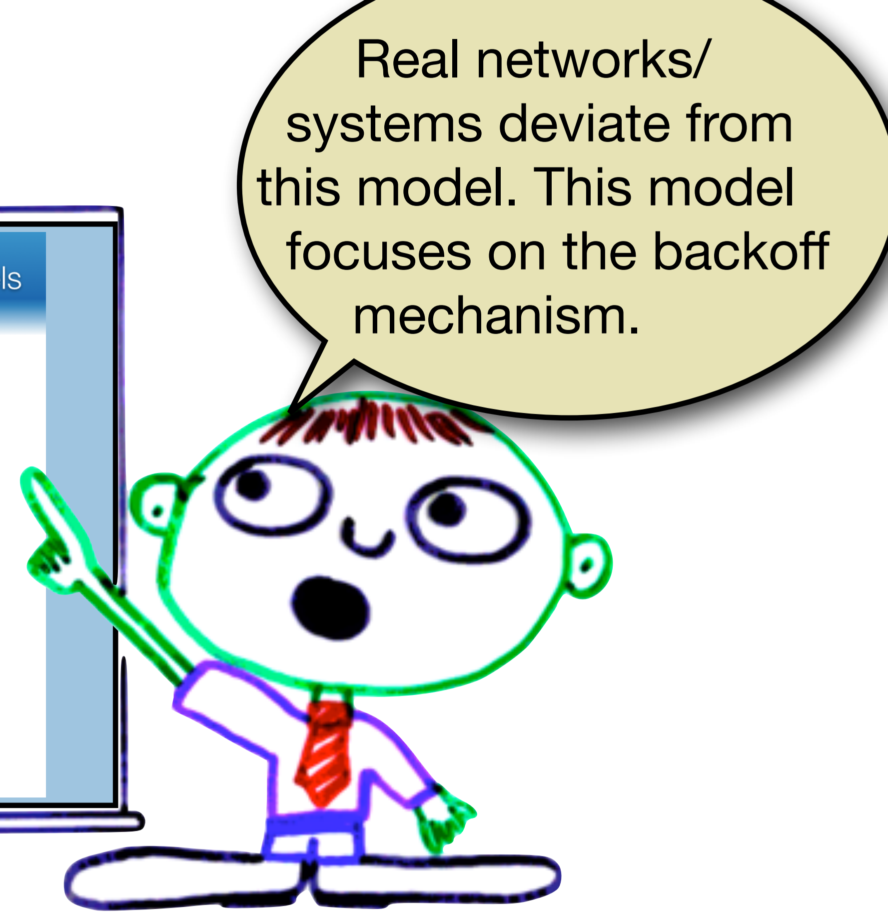


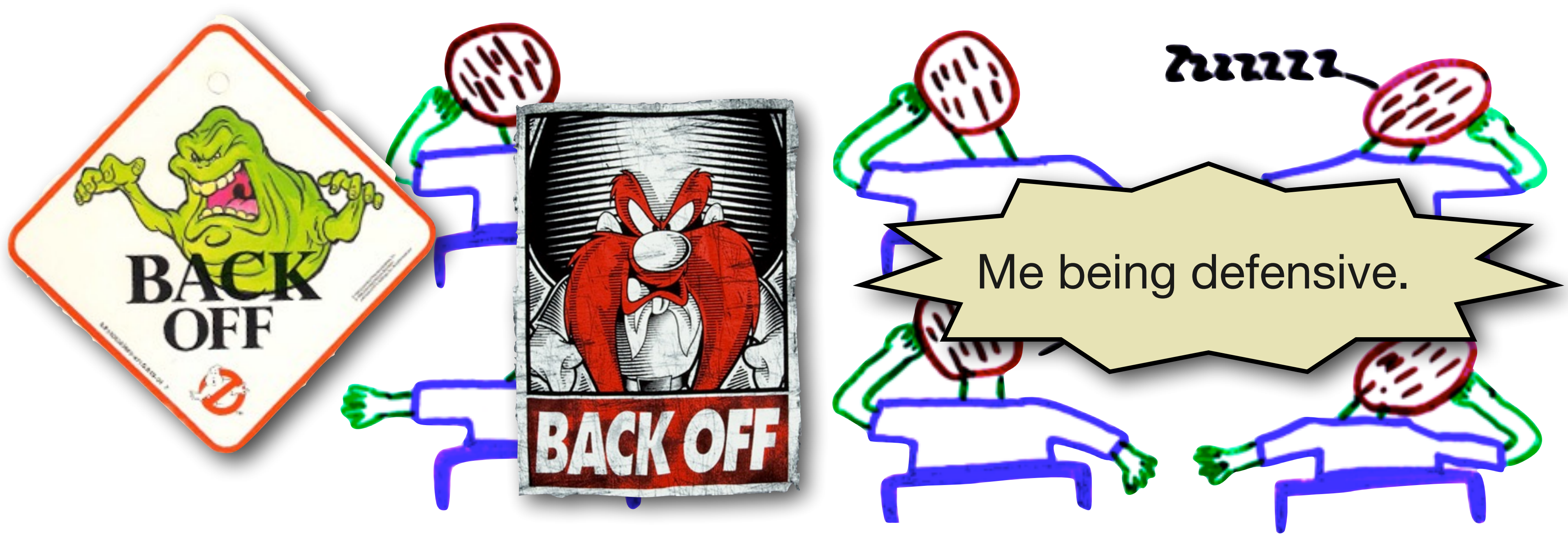
In every slot, a device can:

- **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**.





Binary exponential backoff is broken

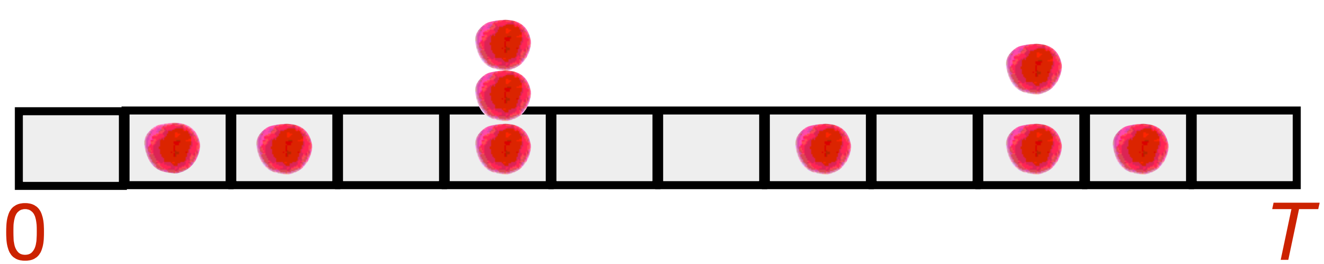
- batch (all release times = 0)
- dynamic arrivals (arbitrary release times)

Batch scenario

All n packets arrive time $t = 0$.

Let makespan $= T$.

Throughput: n/T.



throughput = 4/12

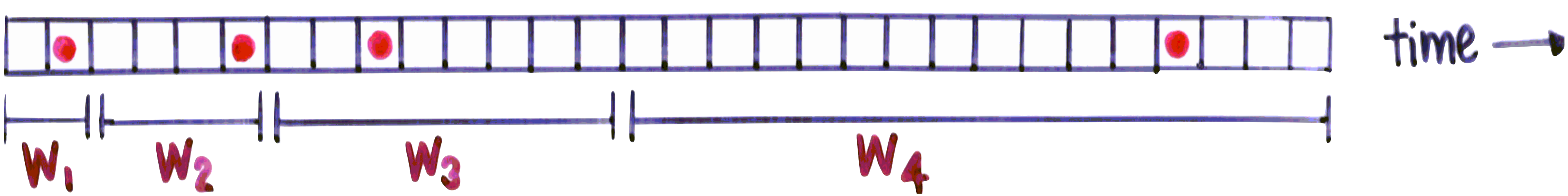
Exponential backoff on batches

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

back off slowly

Additive increase

- After collision: $W = W + 1$

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

back off slowly

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 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?

<u>Approx. running time</u>

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

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

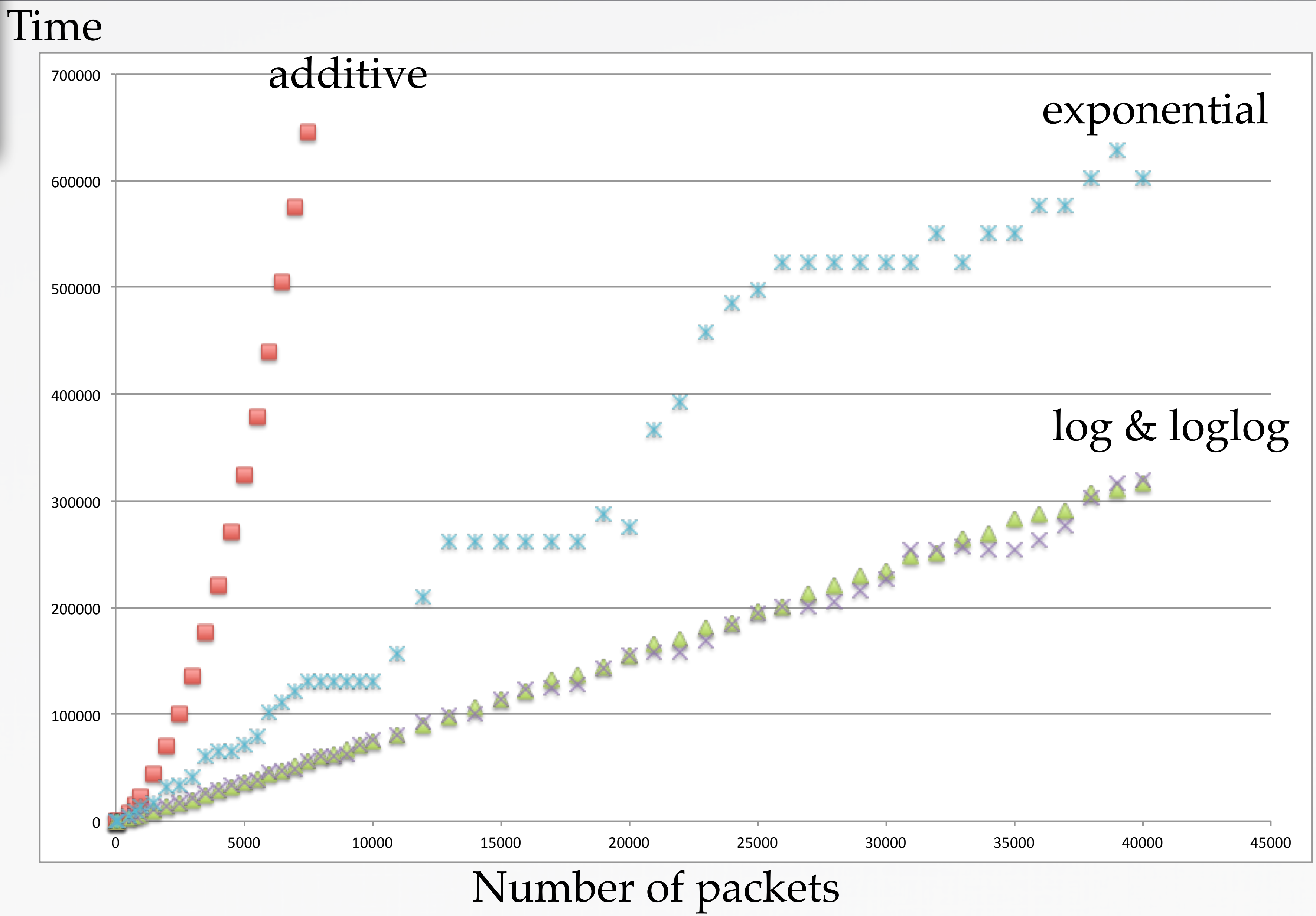
Binary exponential growth

- 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]

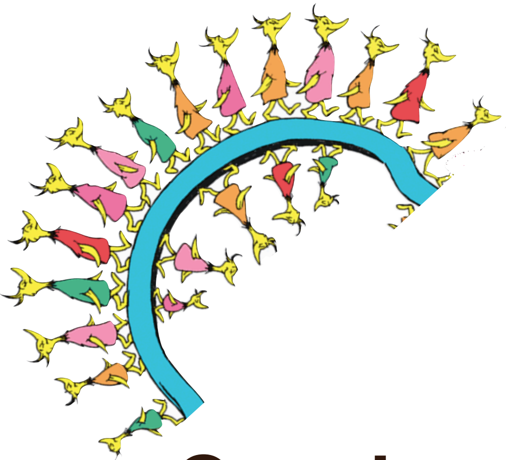
Backoff for batches

Exponential backoff is asymptotically 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.
- But it ***cannot*** achieve a makespan of **O(*n*)** (constant throughput).



Next few slides: dynamic arrivals

(packets have arbitrary release 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

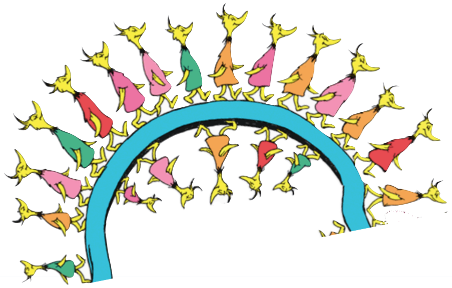
[Bender, Farach-Colton, He, Kuszmaul, Leiserson 05]

- Exponential backoff does not adapt well to bursts.

Adversarial queueing theory with *n* fixed stations

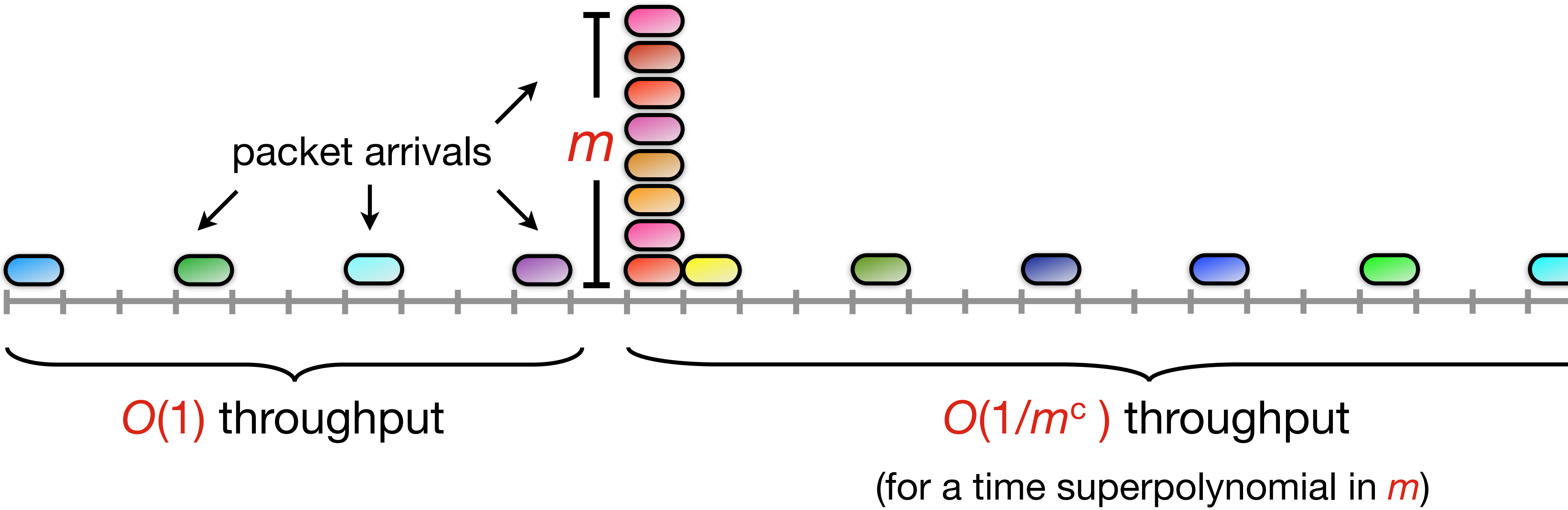
[Chlebus, Kowalski, Rokicki 06 12] [Anantharamu, Chlebus, Rokicki 09] [Chlebus, Kowalski 04] [Chlebus, Gasieniec, Kowalsi, Radzik 05] [Chrobak, Gasieniec, Kowalski 07] etc

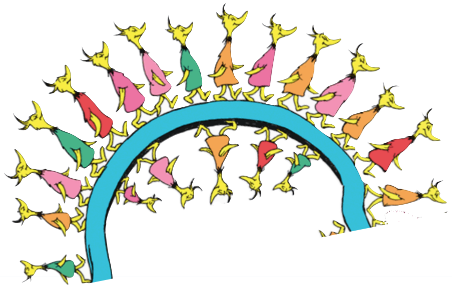
- Adversarial injections
- Often deterministic algorithms: round-robin/binary search/etc.



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]





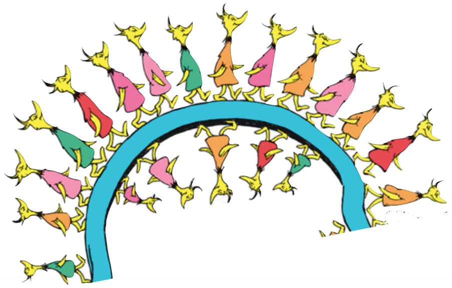
Exponential backoff and bursts

Broadcast probability

- A packet in the system for ***d*** time units broadcasts with probability ***Θ*(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)

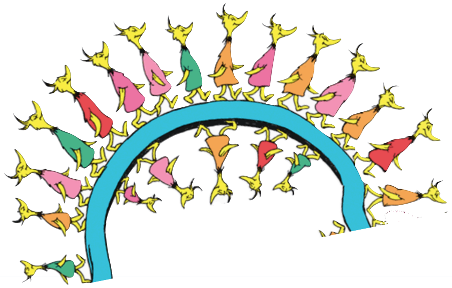
contention c = Ω(1)

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

The success probability is exponentially small in the contention.

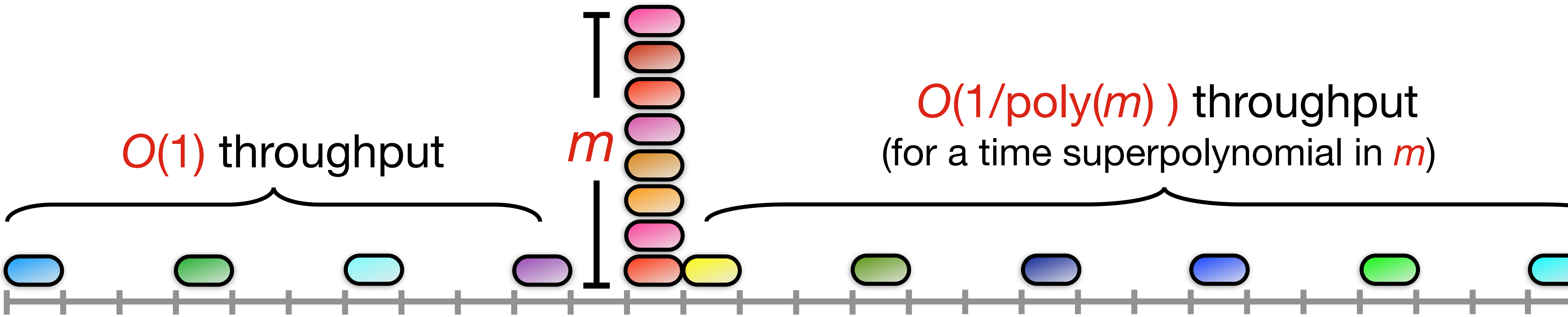
contention c = o(1)

- prob(slot is not empty) = Θ(c)

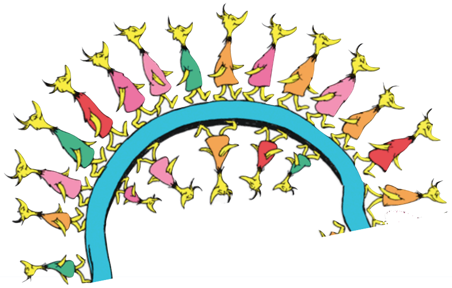


Exponential backoff and bursts

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

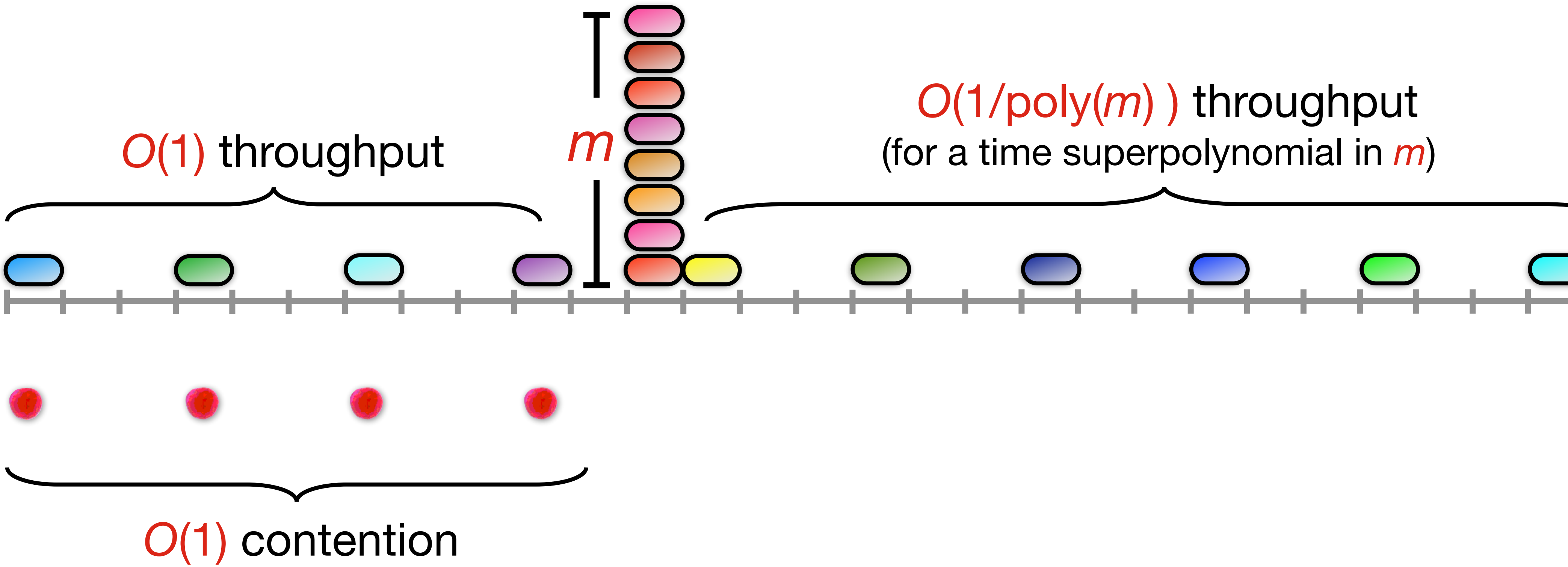


$$O(\log m) = O\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\mathrm{poly}(m)}\right)$$

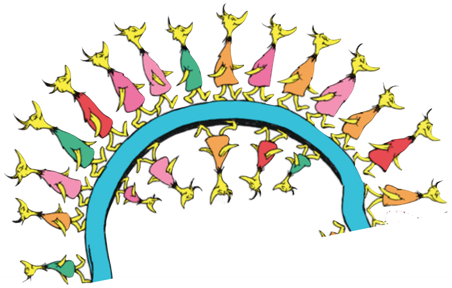


Exponential backoff and bursts

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

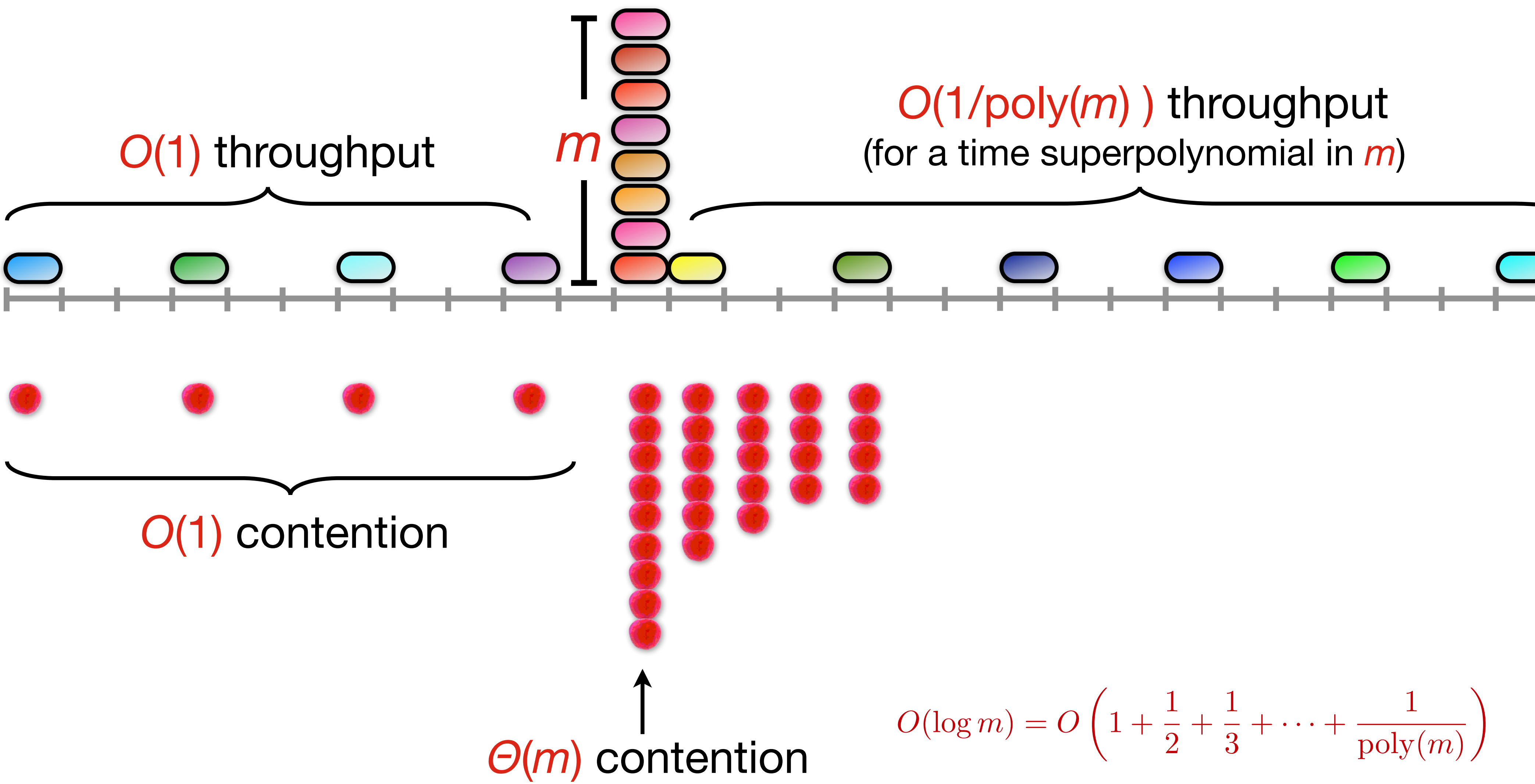


$$O(\log m) = O\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\mathrm{poly}(m)}\right)$$

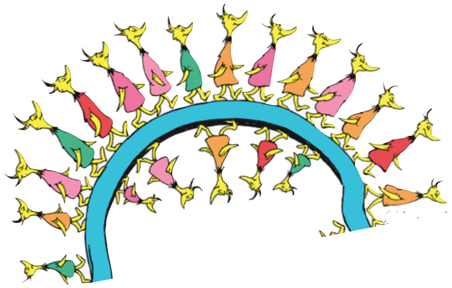


Exponential backoff and bursts

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

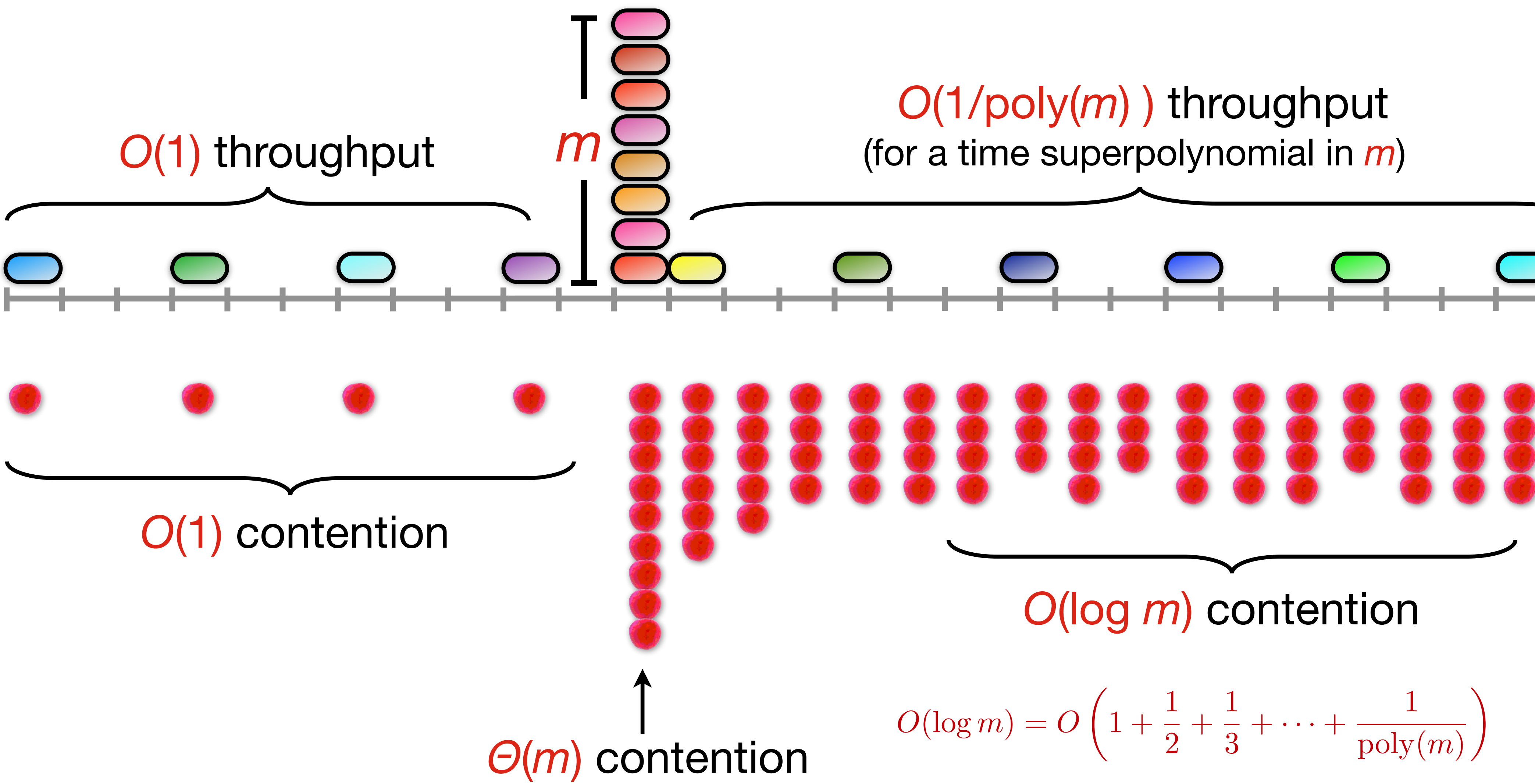


$$O(\log m) = O\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\text{poly}(m)}\right)$$

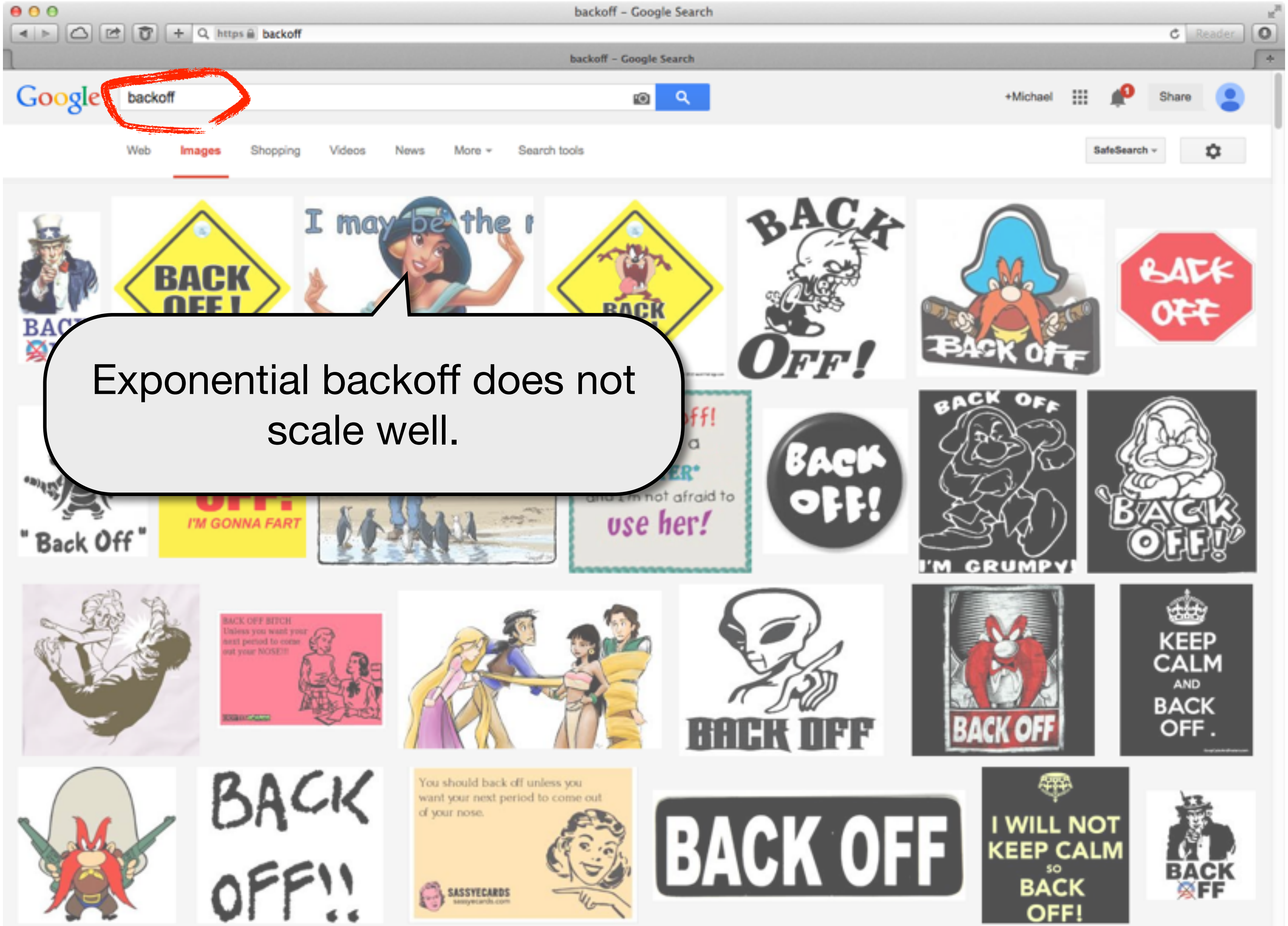


Exponential backoff and bursts

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

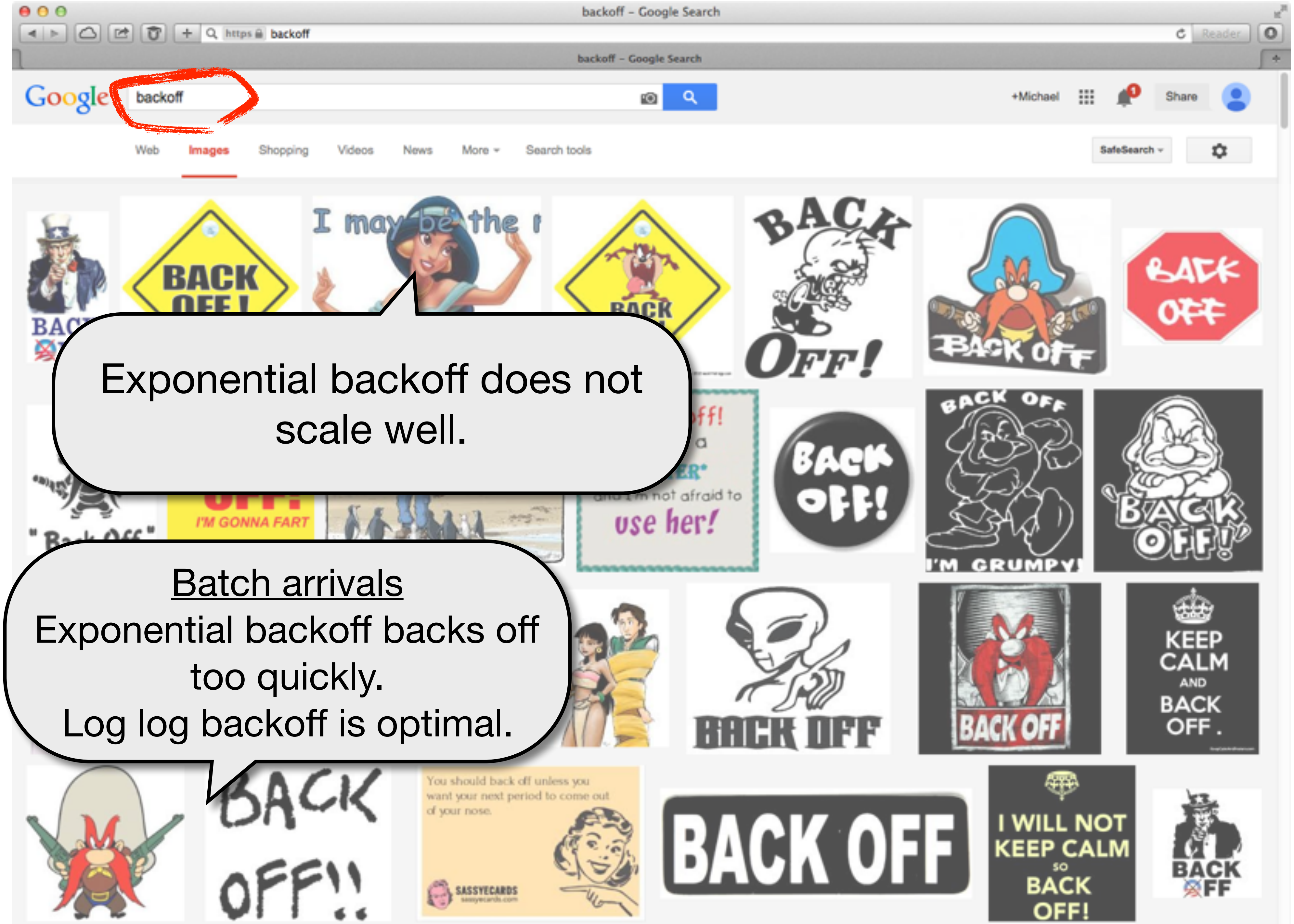


Morals for binary exponential backoff



Exponential backoff does not scale well.

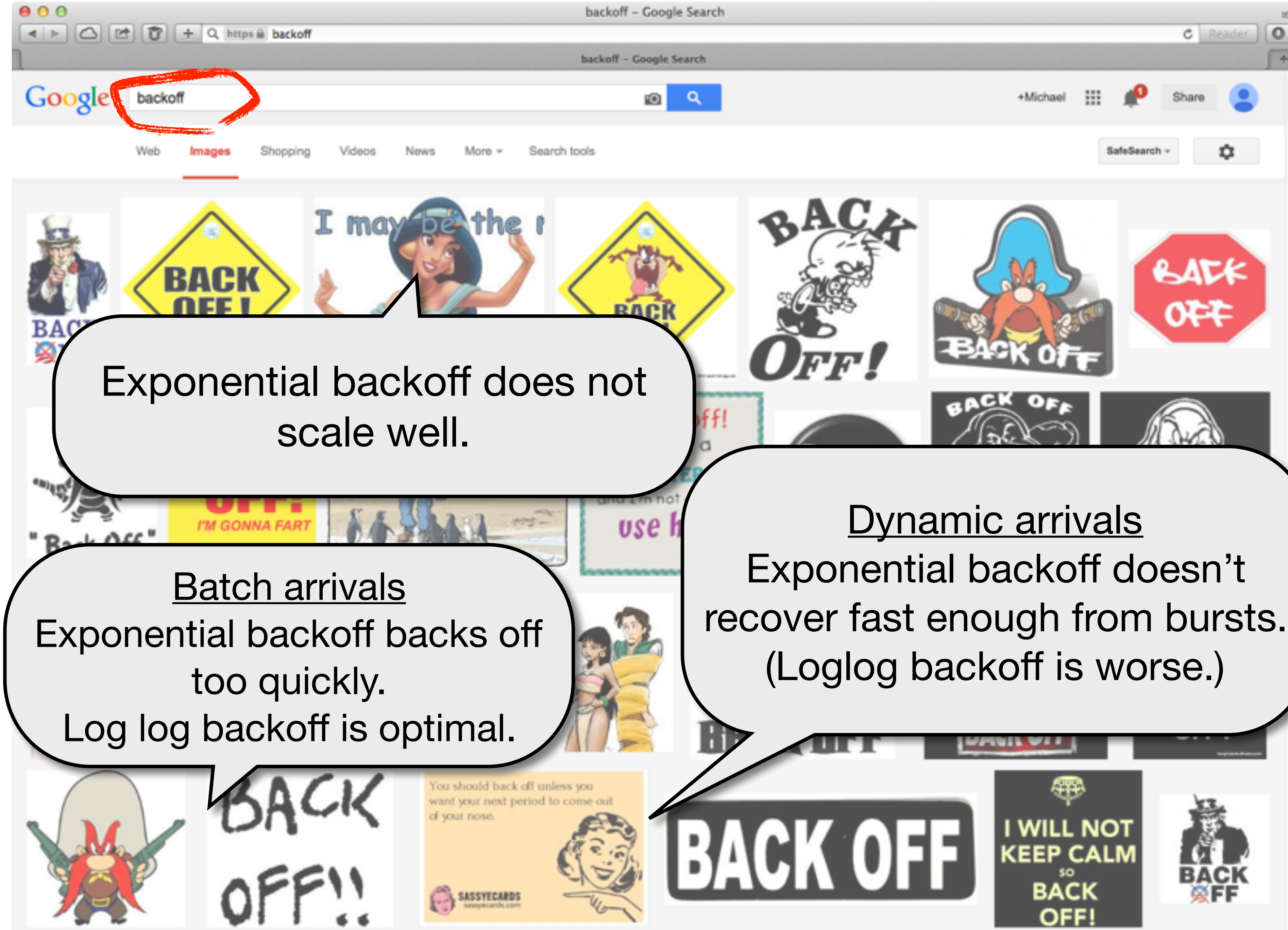
Morals for binary exponential backoff



Exponential backoff does not scale well.

<u>Batch arrivals</u>
Exponential backoff backs off too quickly.
Log log backoff is optimal.

Morals for binary exponential backoff



Exponential backoff does not scale well.

<u>Batch arrivals</u>
Exponential backoff backs off too quickly.
Log log backoff is optimal.

<u>Dynamic arrivals</u>
Exponential backoff doesn't recover fast enough from bursts.
(Loglog backoff is worse.)

How to scale exponential backoff

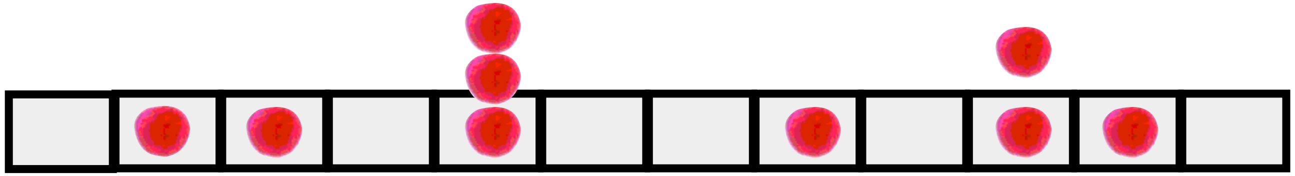
- Analyze batch arrivals (a single burst).
- Analyze dynamic arrivals...
  by reducing to series of batches.
- Good makespan, good # broadcasts

with jamming/failures: [Bender, Fineman, Gilbert, Young, SODA 16]
without jamming: [Bender, Kopelowitz, Pettie, Young, STOC 16]

Batch arrivals

TBD

minimize makespan
minimize effort
achieve robustness to faults and jamming

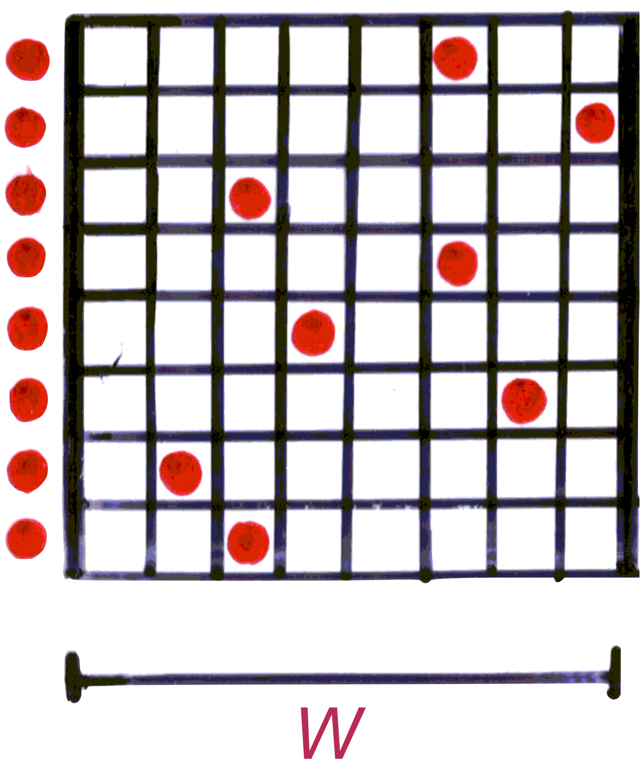


throughput = 4/12

Constant throughput for batches

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

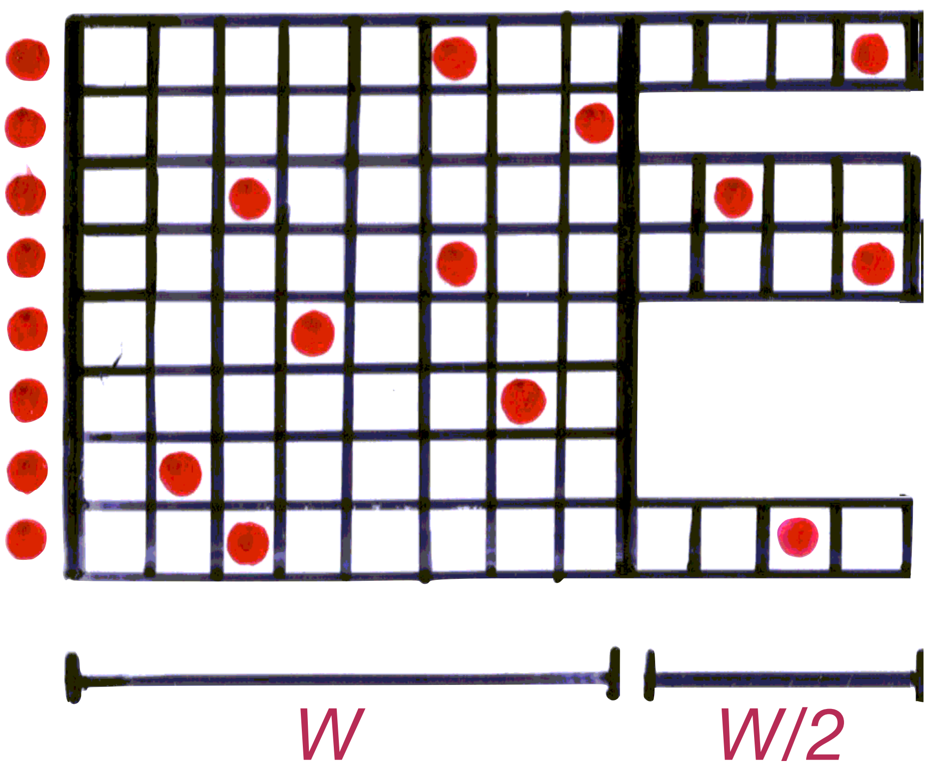
Upshot: We can reduce W by a constant factor



Constant throughput for batches

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

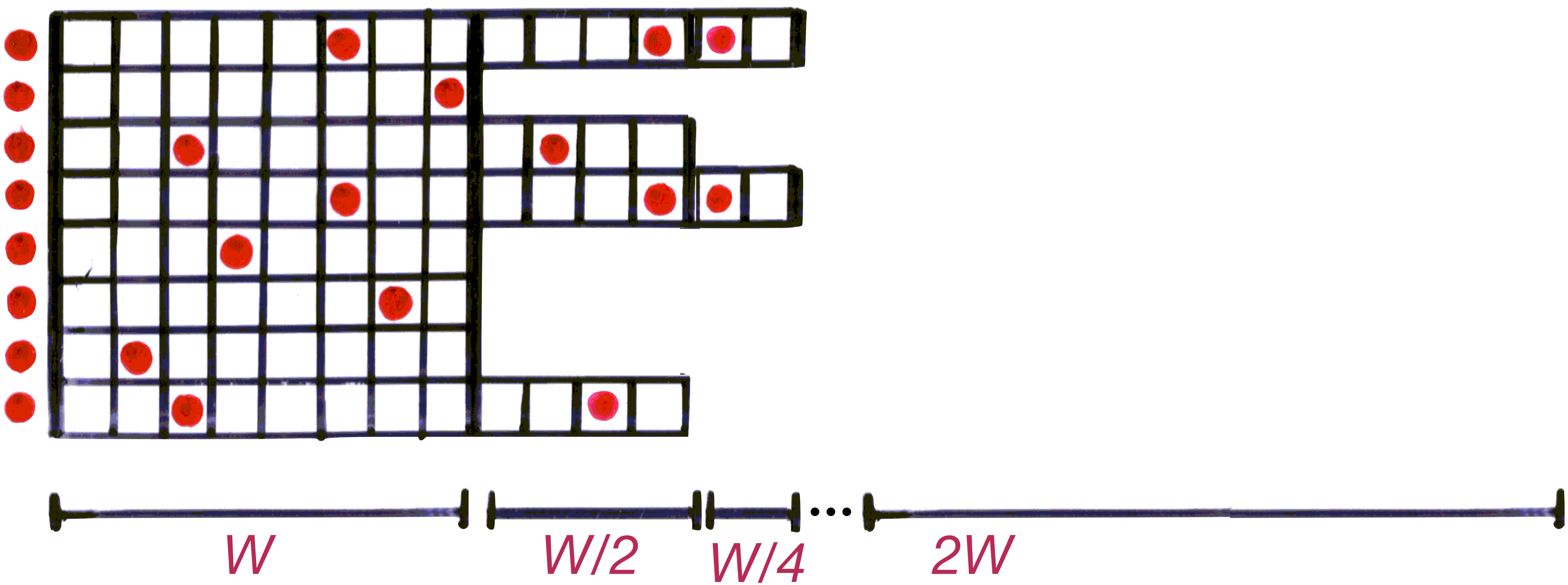
Upshot: We can reduce W by a constant factor.



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.



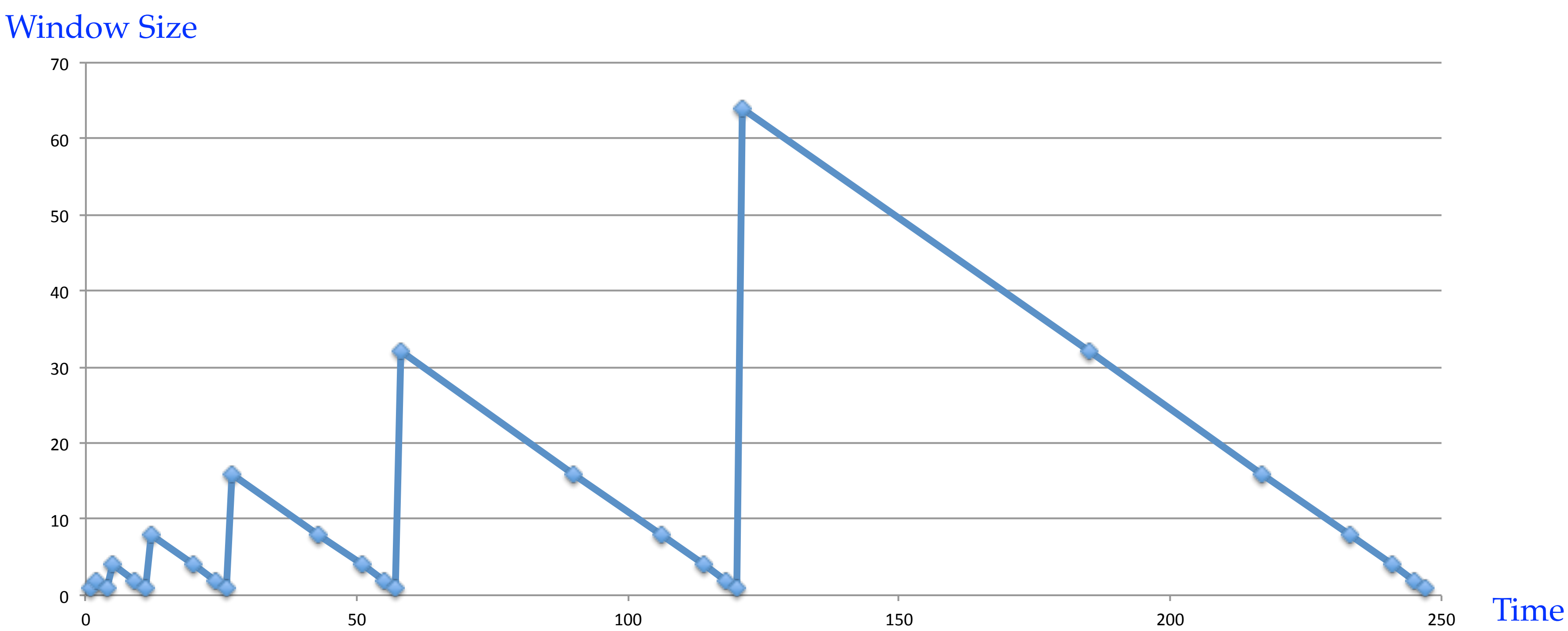
Sawtooth backoff

[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 $W = 2n$.



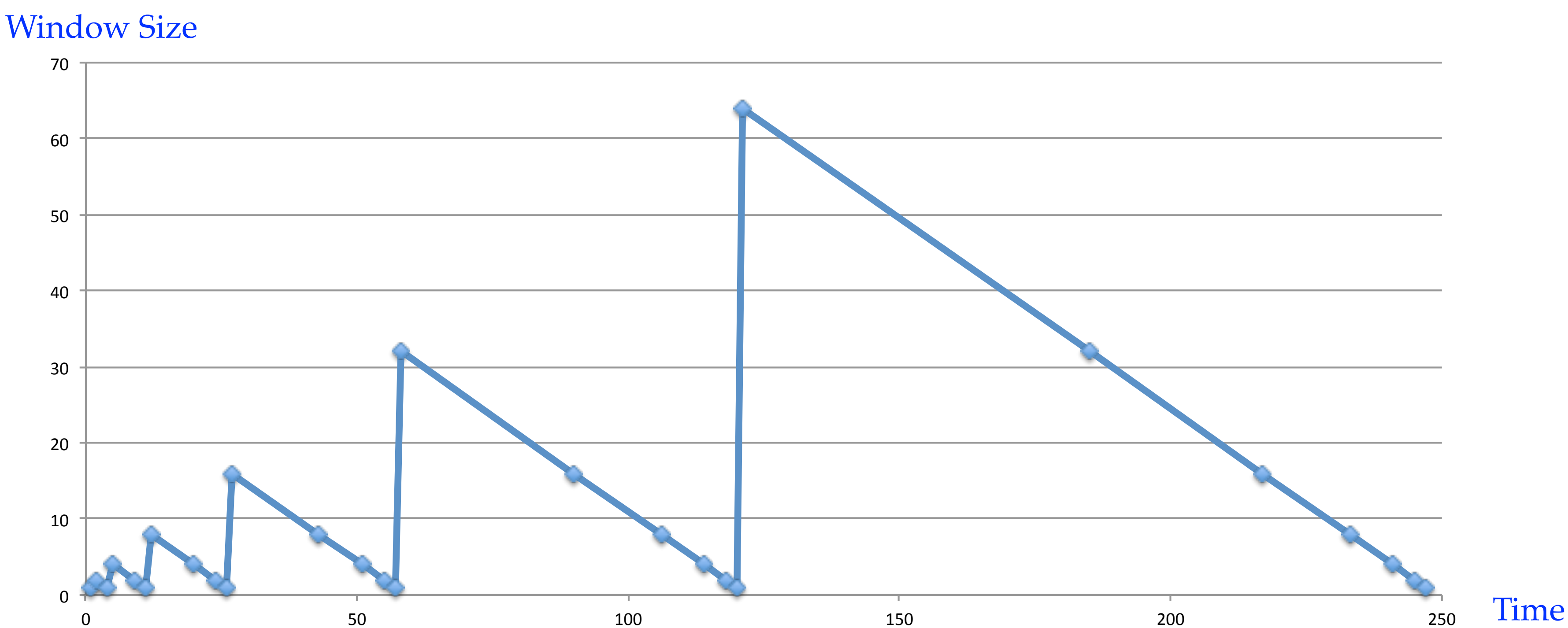
Sawtooth backoff

[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.



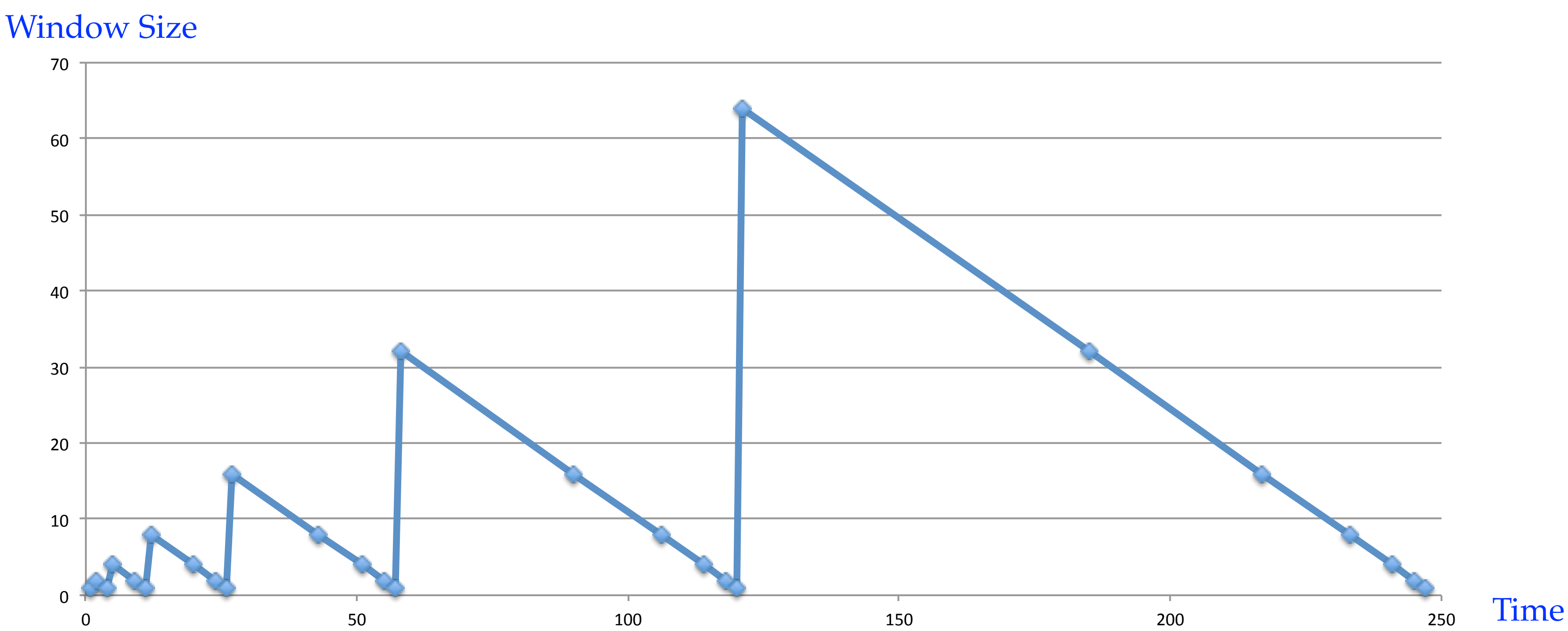
Sawtooth backoff

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

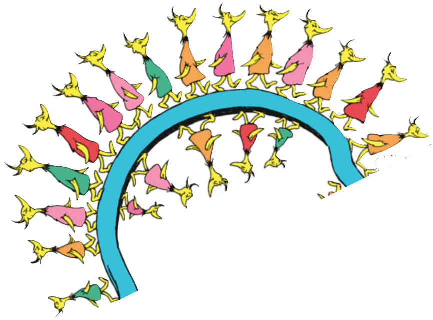
Theorem: For n packet that arrive at time 0, w.h.p., all packets transmit after

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

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



(If we know n, we obtain $O(n)$ makespan with $O(1)$ expected attempts.)



Some Results for Dynamic Arrivals

[Bender, Fineman, GIlbert, Young SODA16]

Theorem:

n = # packets.

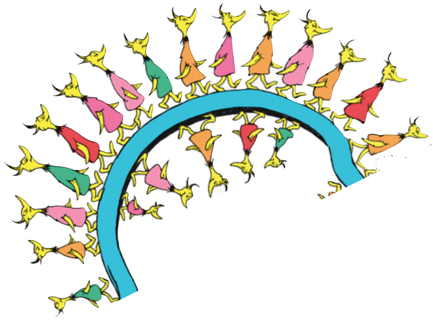
f = # slots blocked by adversary.

makespan: $O(n+f)$ in expectation

- $\Theta(1)$ throughput when $f=O(n)$.

broadcasts: $O(\log^2(n+f))$ in expectation.

(Not complicated algorithm. Complicated analysis.)



Some Results for Dynamic Arrivals

[Bender, Fineman, GIlbert, Young SODA16]

Theorem:

n = # packets.

f = # slots blocked by adversary.

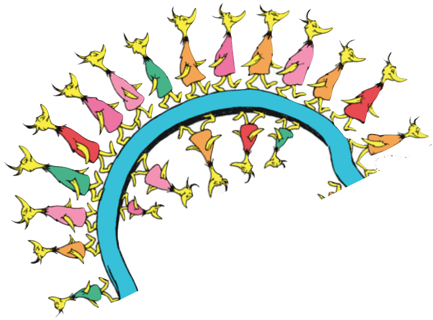
makespan: $O(n+f)$ in expectation

- $\Theta(1)$ throughput when $f=O(n)$.

broadcasts: $O(\log^2(n+f))$ in expectation.

(We think listening can also be optimized, but that's not what this paper is above.)

(Not complicated algorithm. Complicated analysis.)



Some Results for Dynamic Arrivals

[Bender, Kopelowitz, Pettie, Young, STOC16]

Theorem:

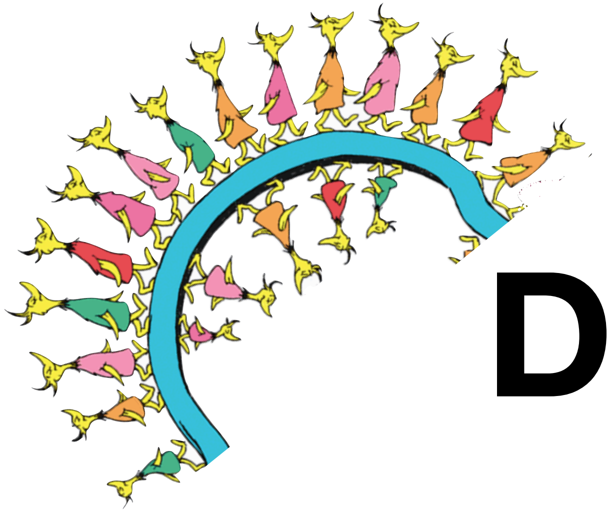
n = # packets.

no jamming of slots.

makespan: $O(n)$ in expectation.

channel accesses: $O(\log \log^* n)$ in expectation.

(Complicated algorithm and analysis.)



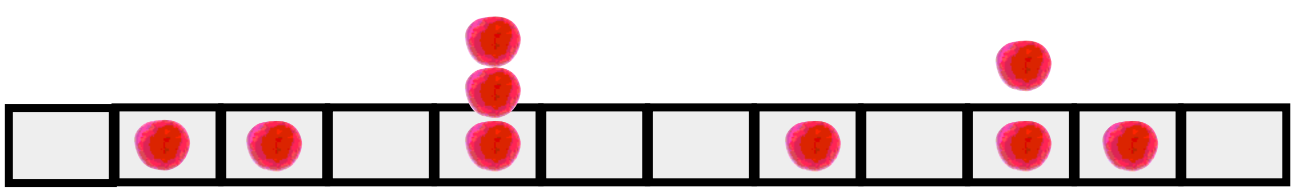
Dynamic arrivals

maximize throughput

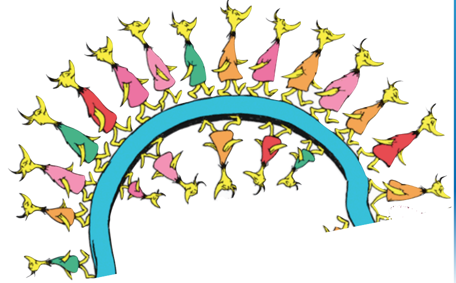
minimize effort

achieve robustness

[Bender, Fineman, GIlbert, Young SODA16]

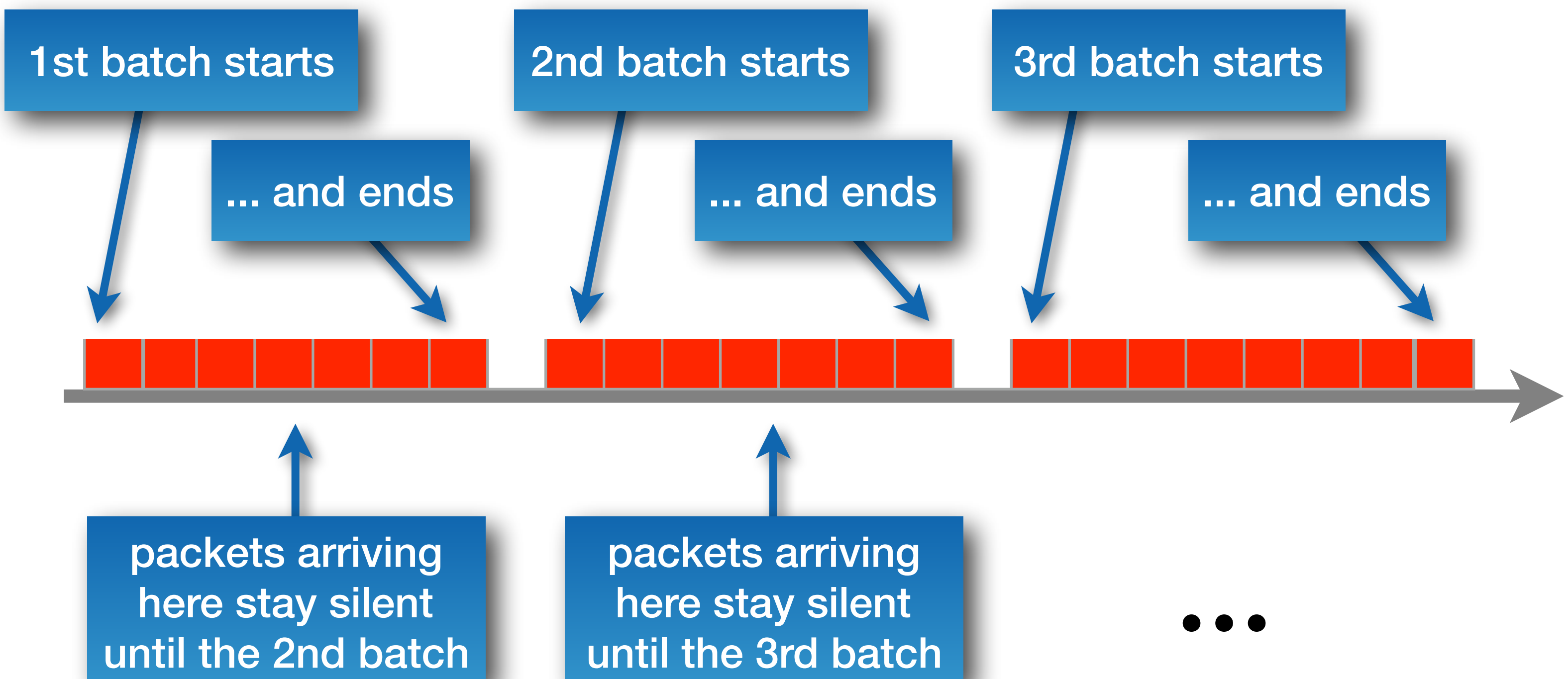


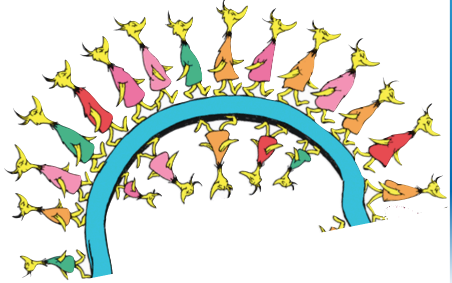
throughput = 4/12



Dynamic arrivals: synchronize into batches

Group packets into synchronized batches.

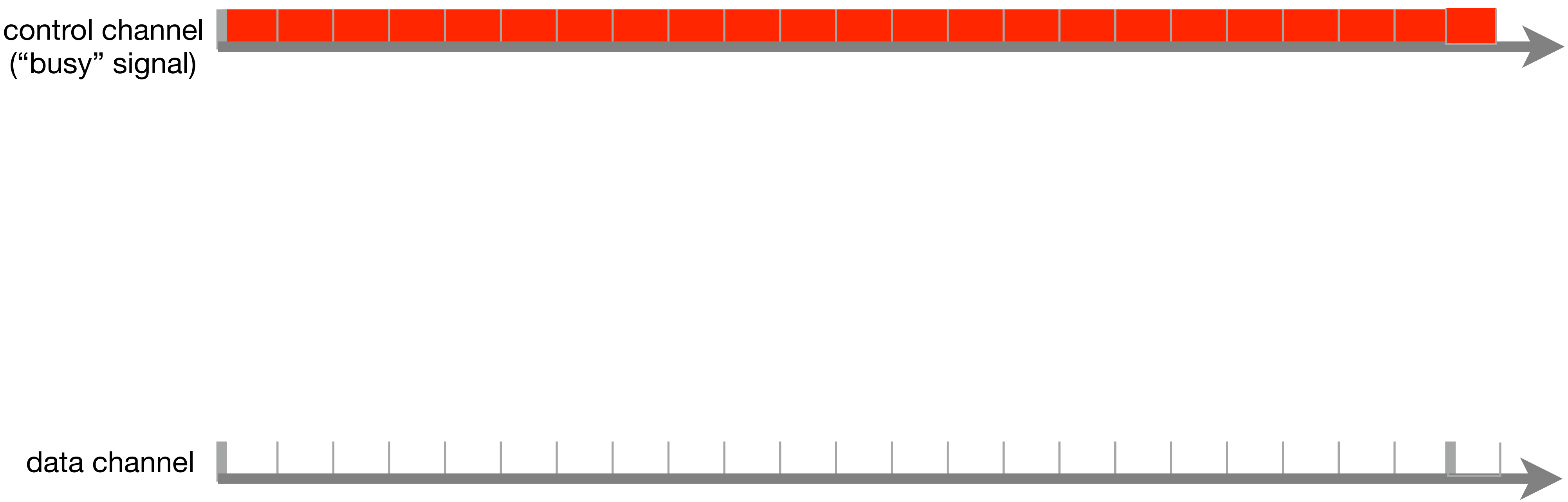


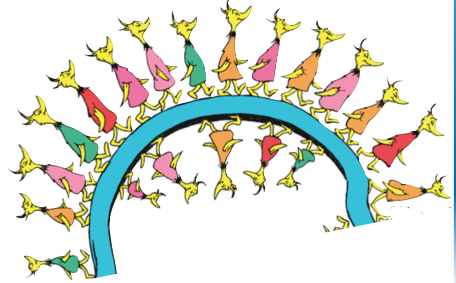


Use two channels (simulate on one)

Assume two channels.

We use the 2nd channel to synchronize into batches.

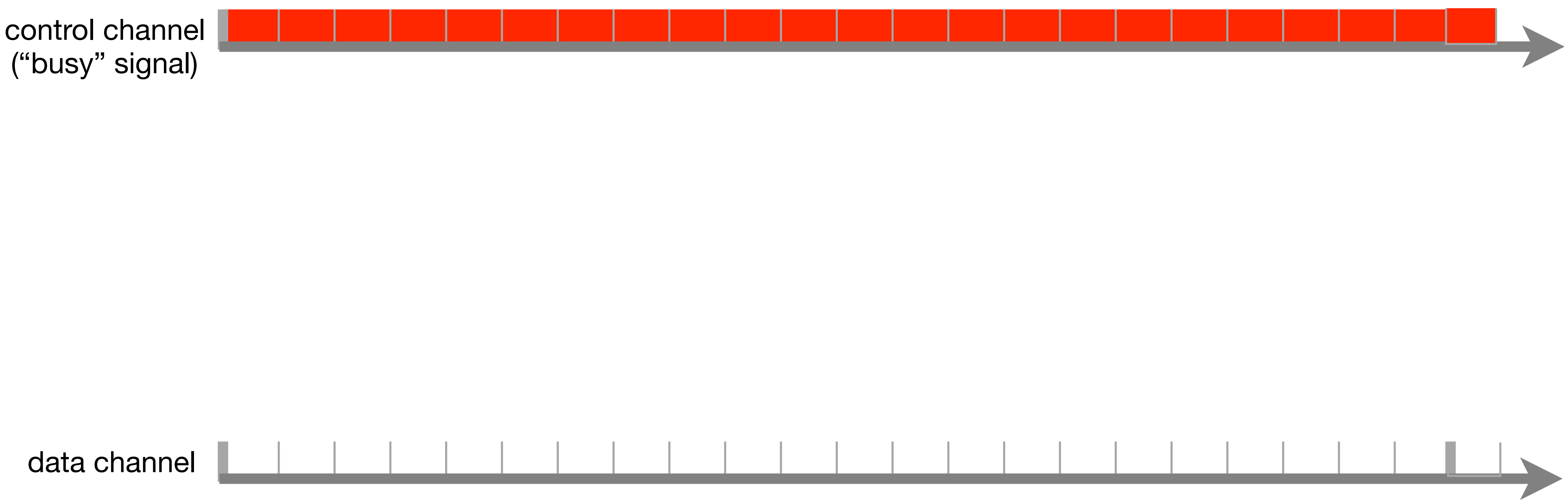




Use two channels (simulate on one)

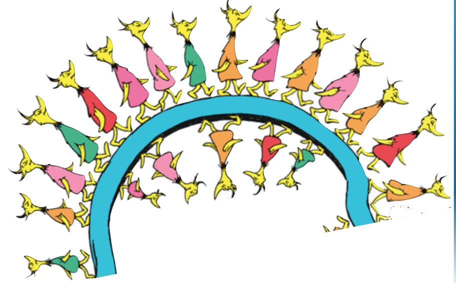
Assume two channels.

We use the 2nd channel to synchronize into batches.



We can simulate two channels on one.

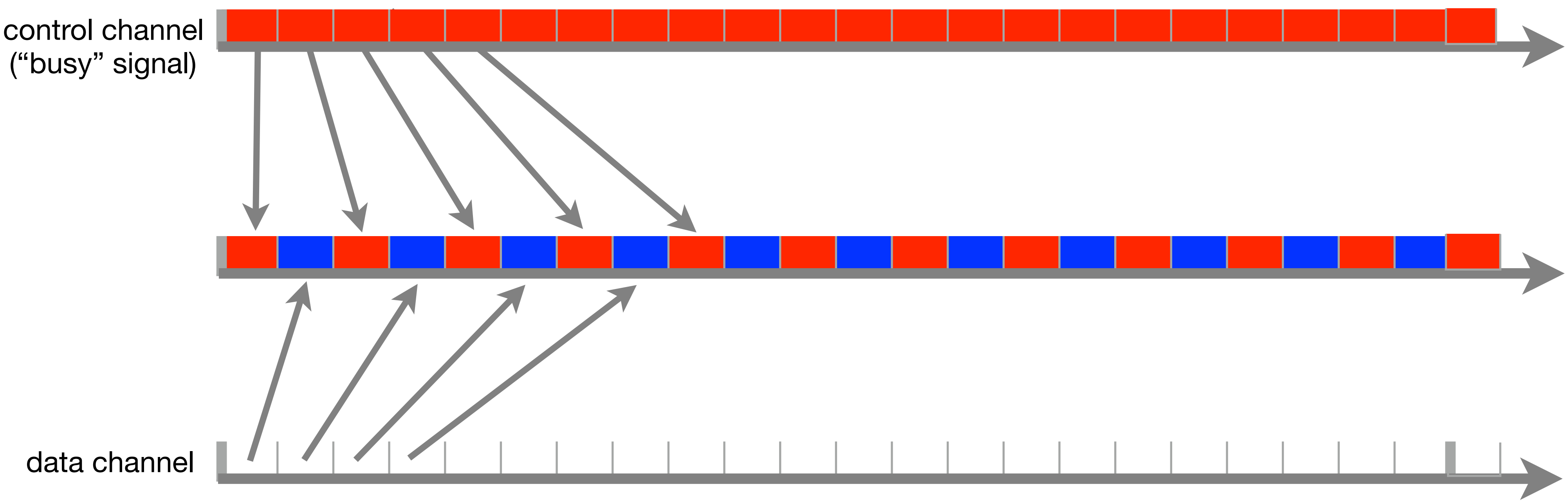
One assumption: *even/odd* round parity is known. Can be dispensed with as well.



Use two channels (simulate on one)

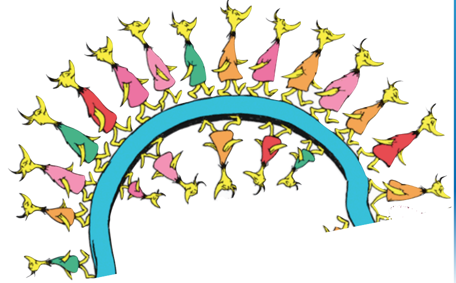
Assume two channels.

We use the 2nd channel to synchronize into batches.



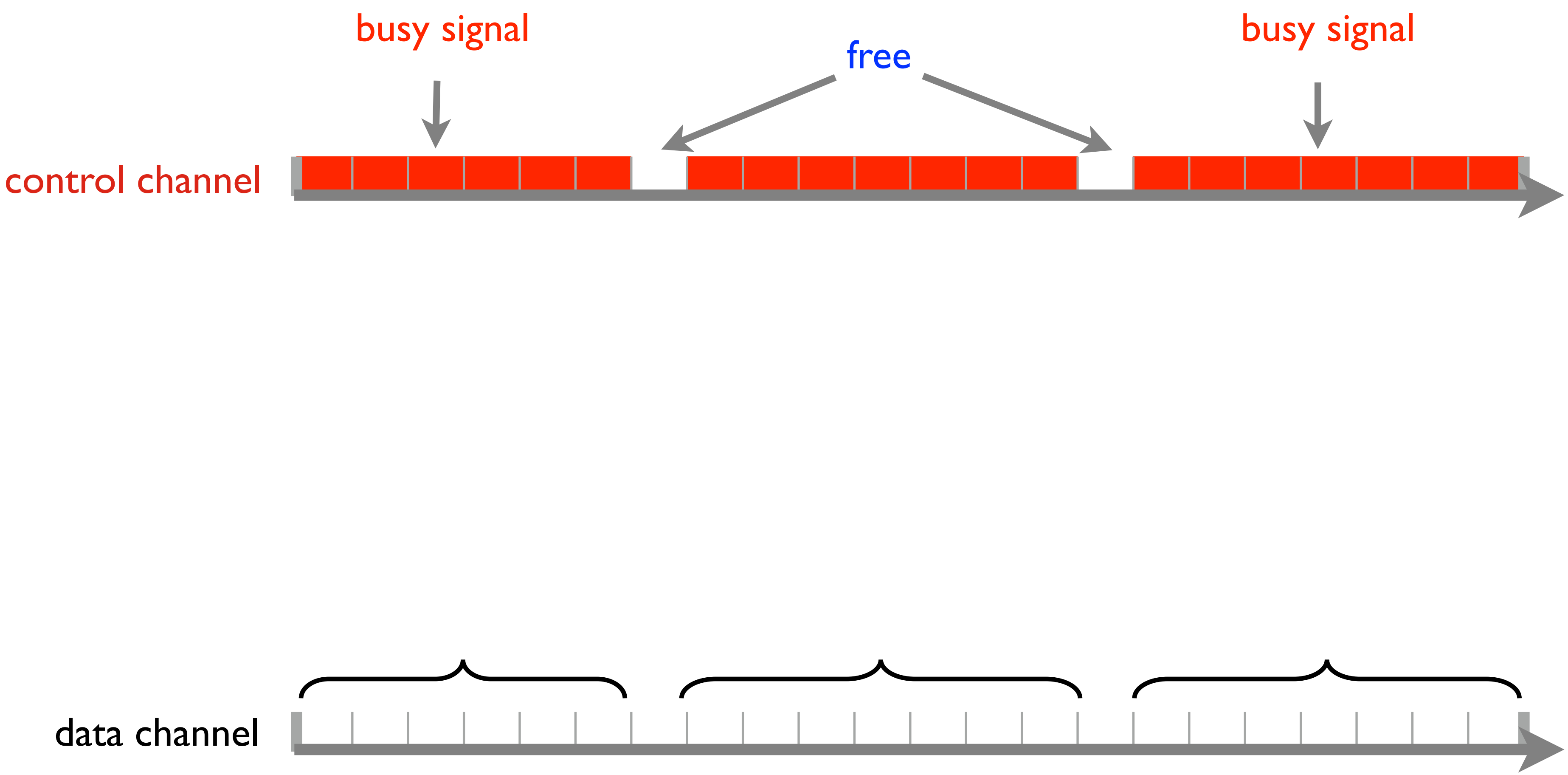
We can simulate two channels on one.

One assumption: *even/odd* round parity is known. Can be dispensed with as well.

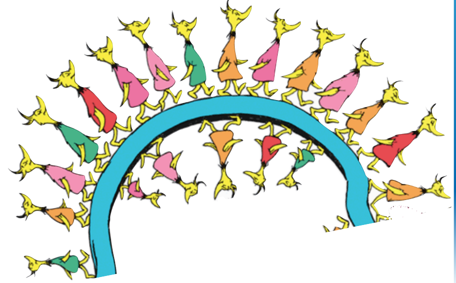


Synchronize batches using busy signal

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

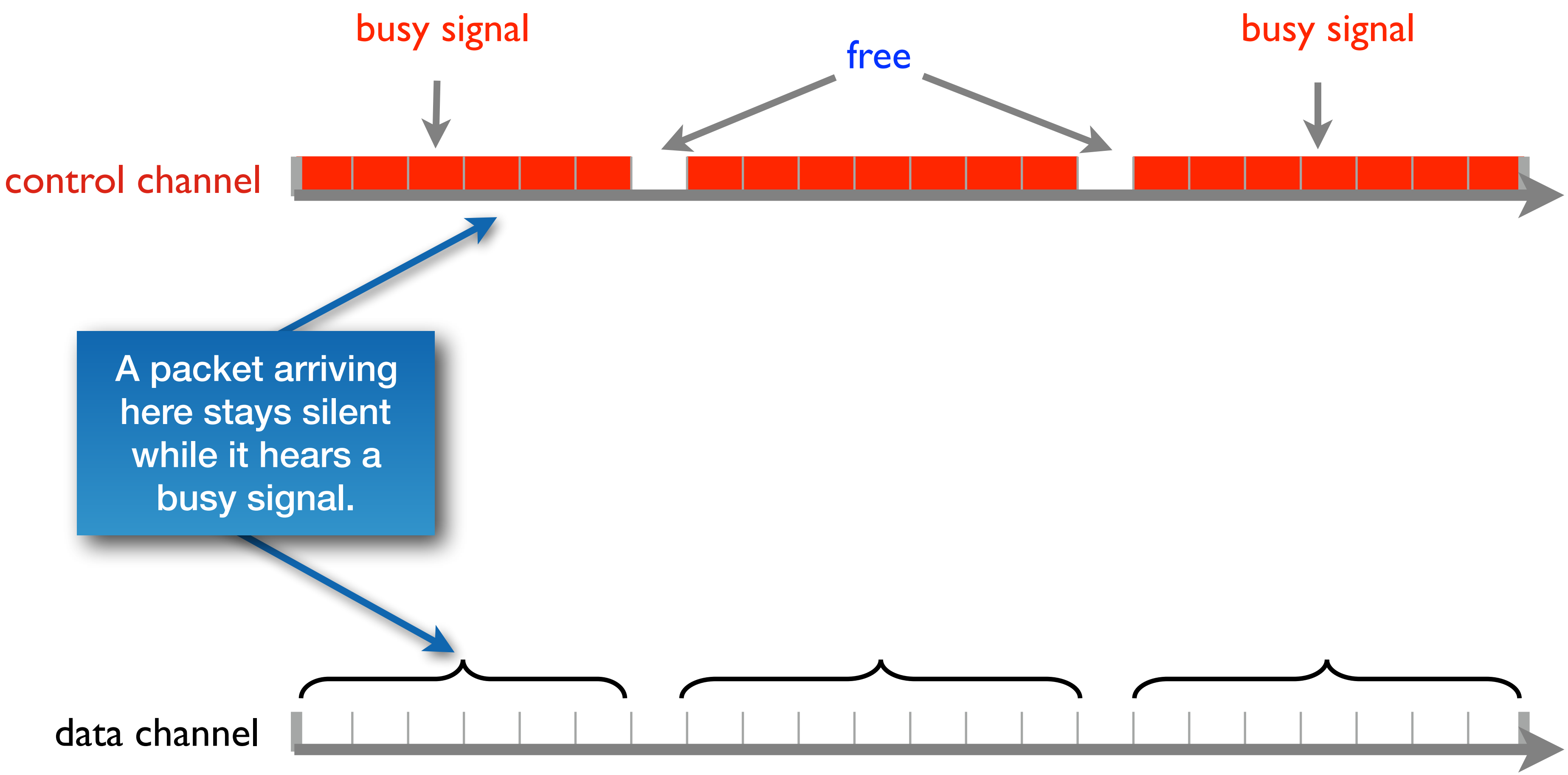


Data channel implements batches.

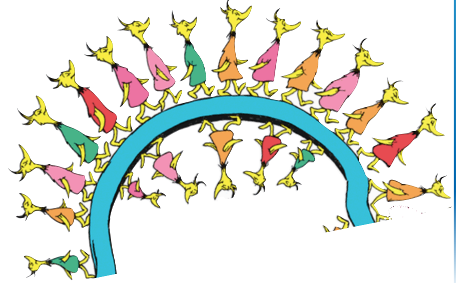


Synchronize batches using busy signal

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

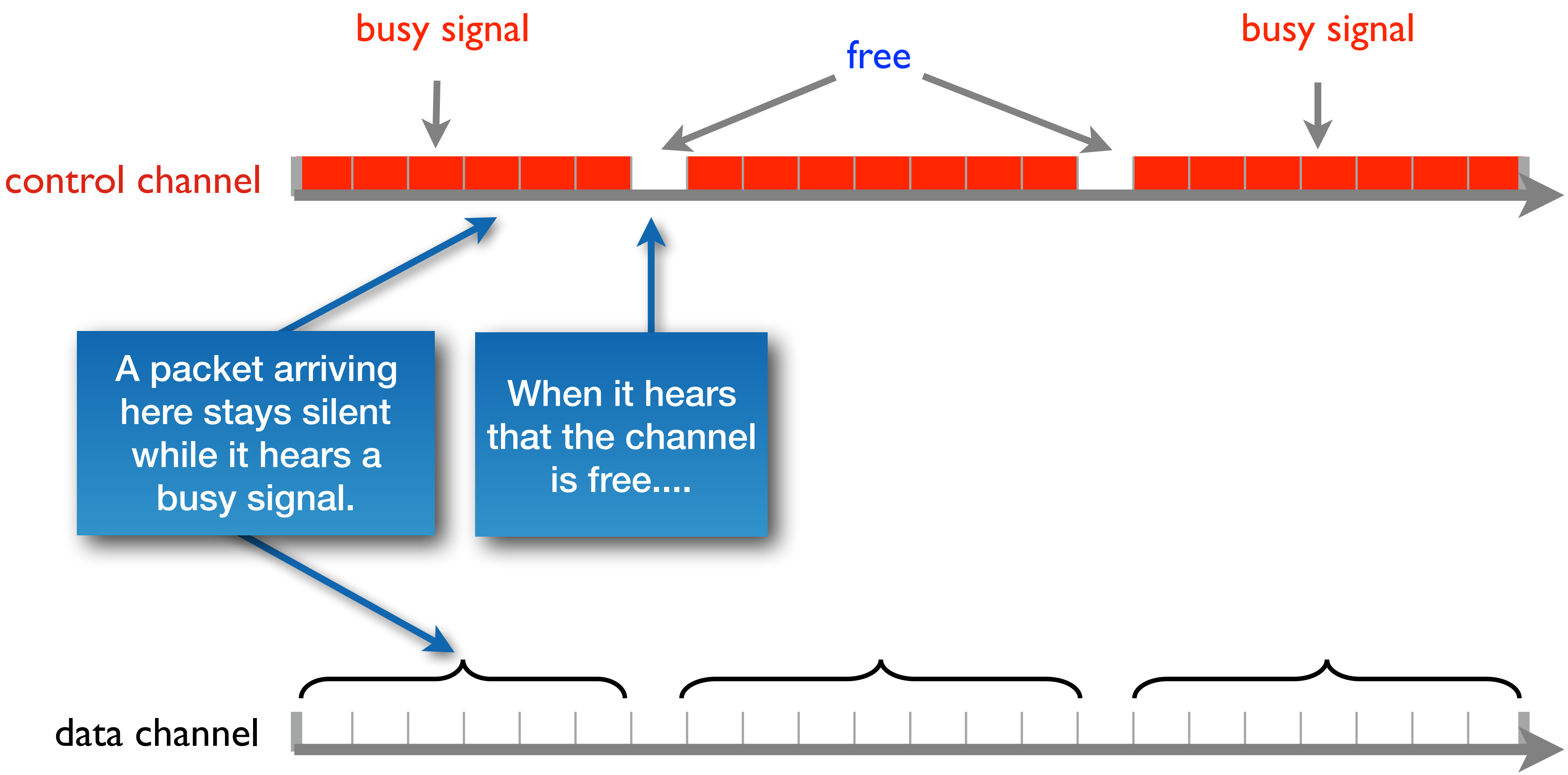


Data channel implements batches.

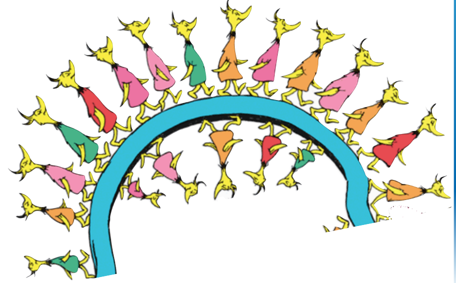


Synchronize batches using busy signal

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

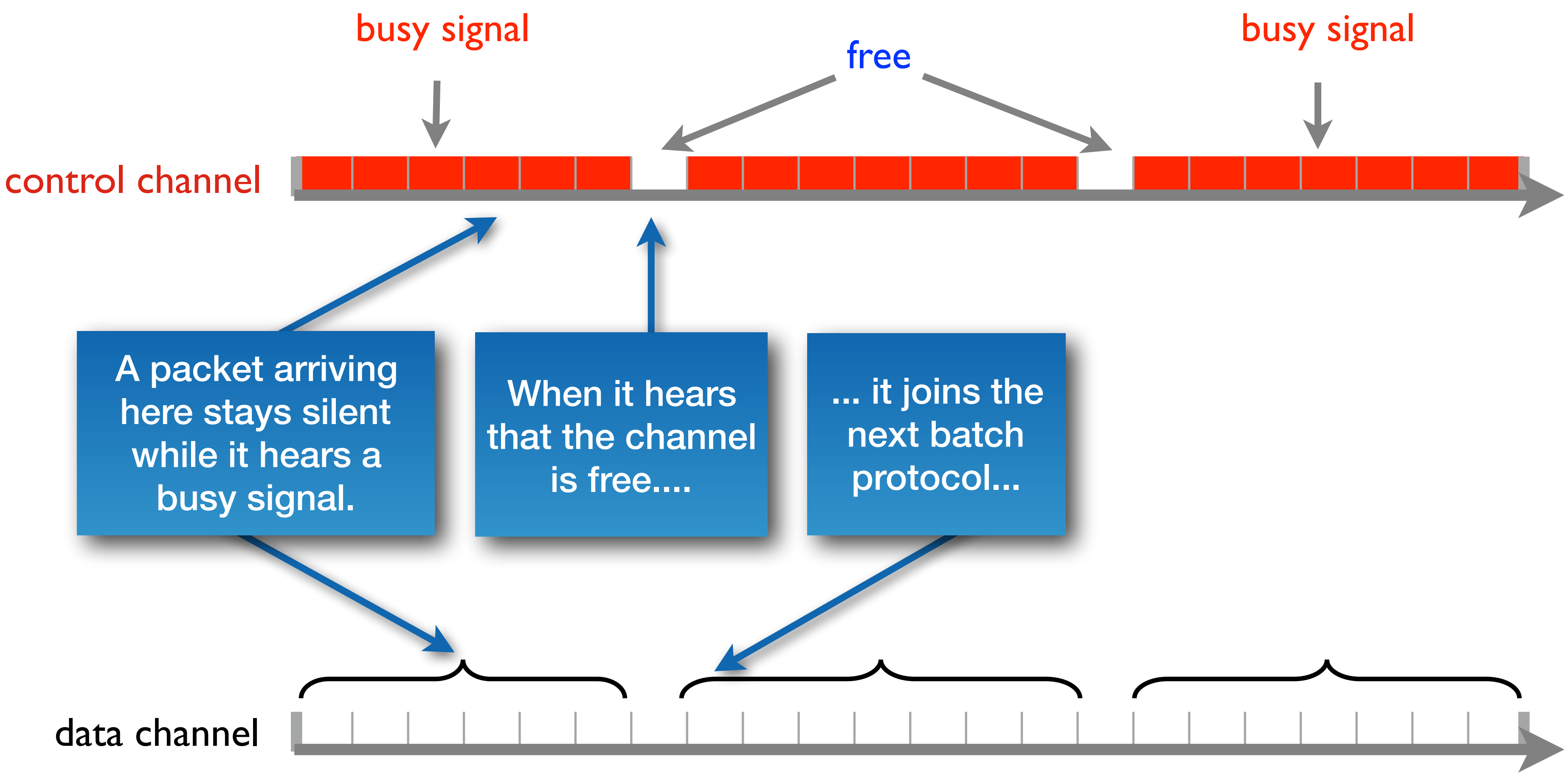


Data channel implements batches.

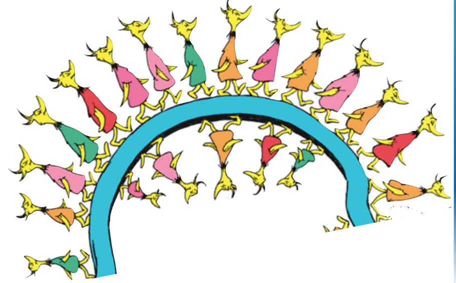


Synchronize batches using busy signal

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

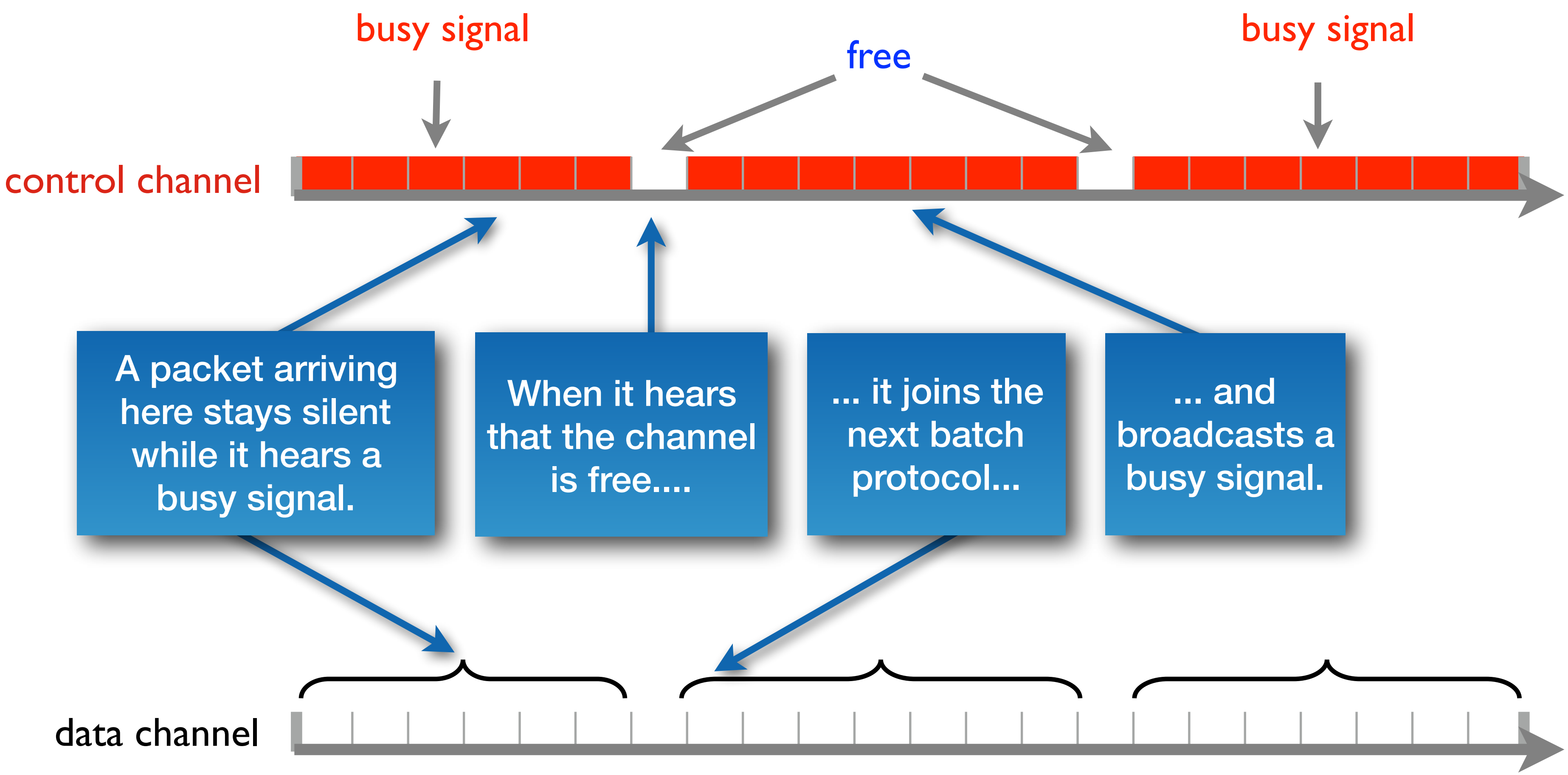


Data channel implements batches.

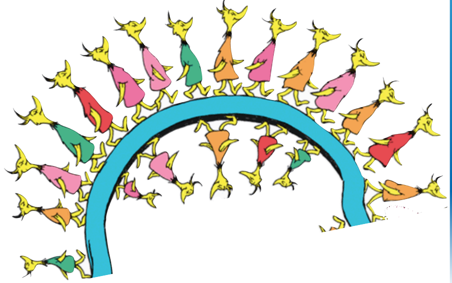


Synchronize batches using busy signal

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



Data channel implements batches.



Protocol on one channel

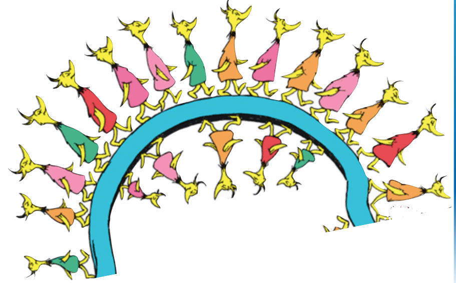
[Bender, Fineman, Gilbert, Young 16]

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 Θ(1) throughput, w.h.p.



Protocol on one channel

[Bender, Fineman, GIlbert, Young 16]

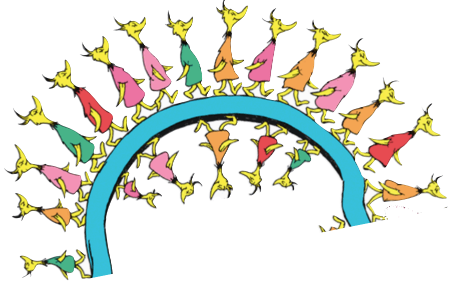
Wait until two consecutive "silent" rounds.

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.

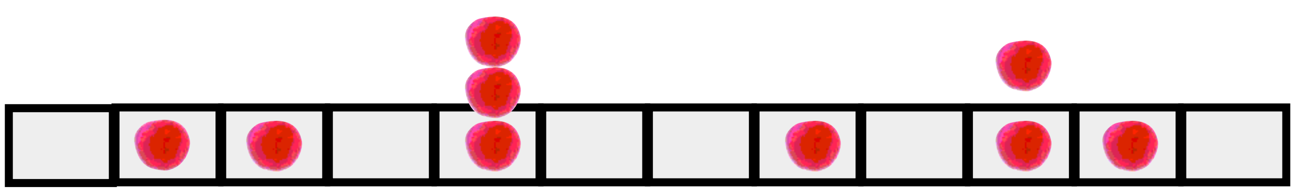


Dynamic arrivals

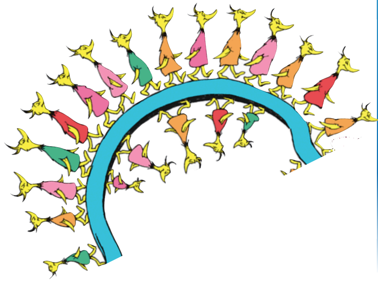
TBD

[Bender, Fineman, GIlbert, Young SODA 16]

maximize throughput
minimize effort
achieve robustness to jamming

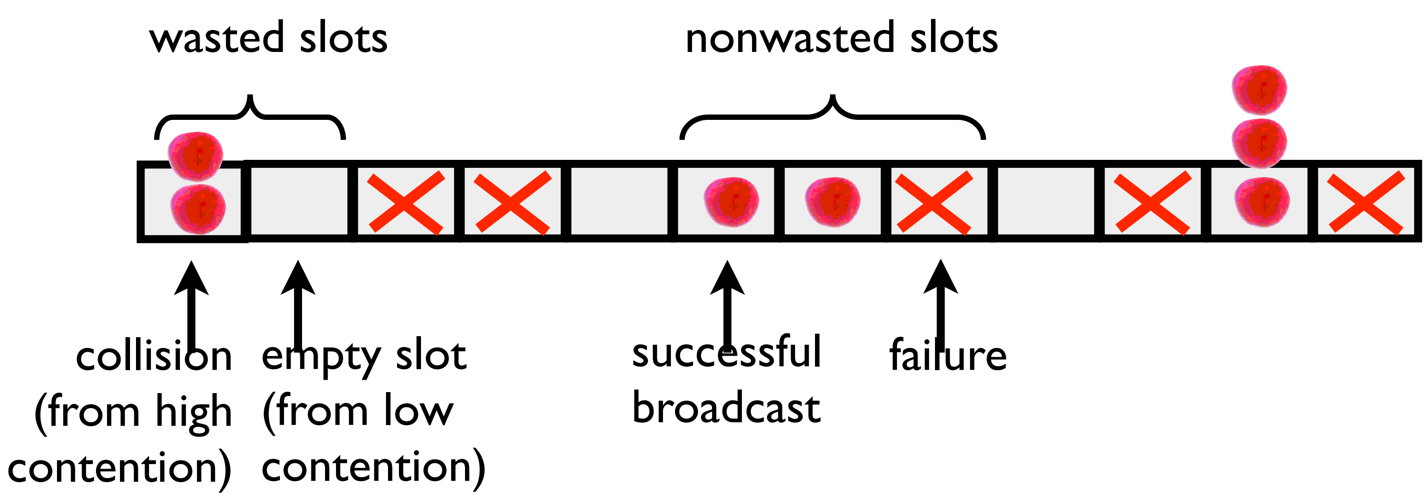


throughput = 4/12



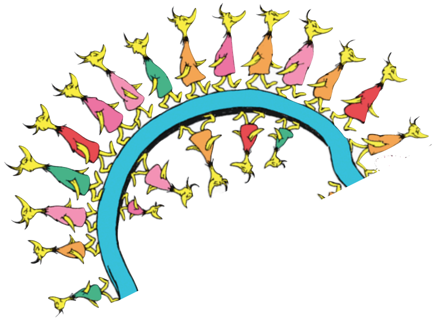
It’s all about contention

Goal: waste O(1) fraction of slots.



Goal: achieve Θ(1) contention on a constant fraction of all slots without doing too many broadcasts.

(Recall: contention = sum of broadcast probabilities.)



Dynamic Arrivals with Jamming

[Bender, Fineman, GIlbert, Young SODA16]

Theorem:

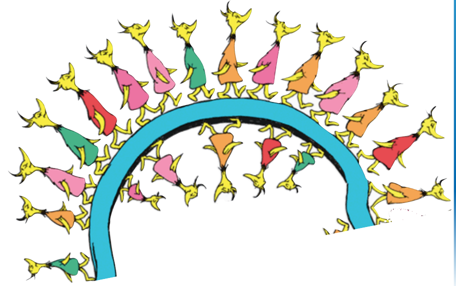
n = # packets.

f = # slots blocked by adversary.

makespan: $O(n+f)$ in expectation

- $\Theta(1)$ throughput when $f=O(n)$.

broadcasts: $O(\log^2(n+f))$ in expectation.



Resolving TDB

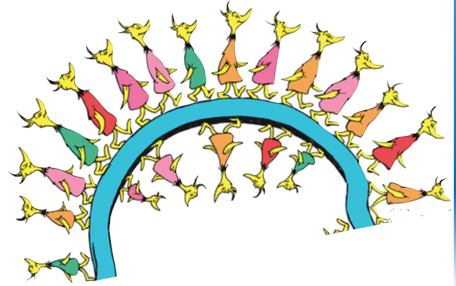
For a packet that's been ***active*** for t slots:

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

For an ***inactive*** request:

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

(Not complicated algorithm. Complicated analysis.)



Resolving TDB

Cheap probabilistic busy signal.

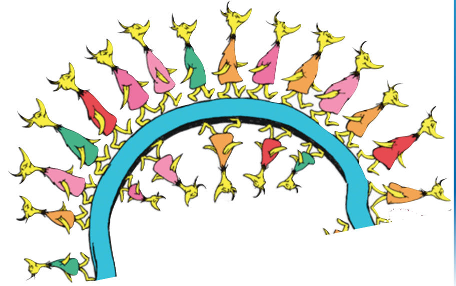
For a packet that's been ***active*** for t slots:

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

For an ***inactive*** request:

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

(Not complicated algorithm. Complicated analysis.)



Resolving TDB

For a packet that's been ***active*** for *t* slots:

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

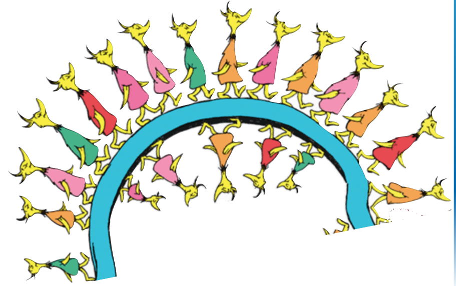
Cheap probabilistic busy signal.

Just like exponential backoff.

For an ***inactive*** request:

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

(Not complicated algorithm. Complicated analysis.)



Resolving TDB

For a packet that's been ***active*** for t slots:

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

Cheap probabilistic busy signal.

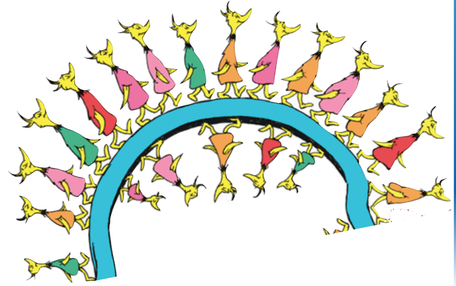
Just like exponential backoff.

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***.

(Not complicated algorithm. Complicated analysis.)



Resolving TDB

For a packet that's been ***active*** for t slots:

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

Cheap probabilistic busy signal.

Just like exponential backoff.

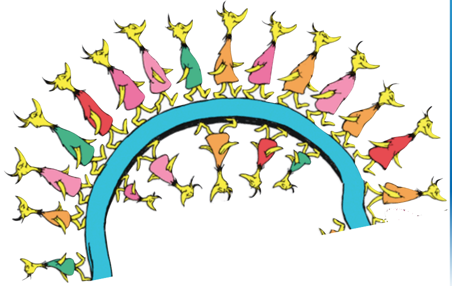
Fault-tolerant measure of low contention.
A batch ends when O(1) fraction of packets finished.

For an ***inactive*** request:

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

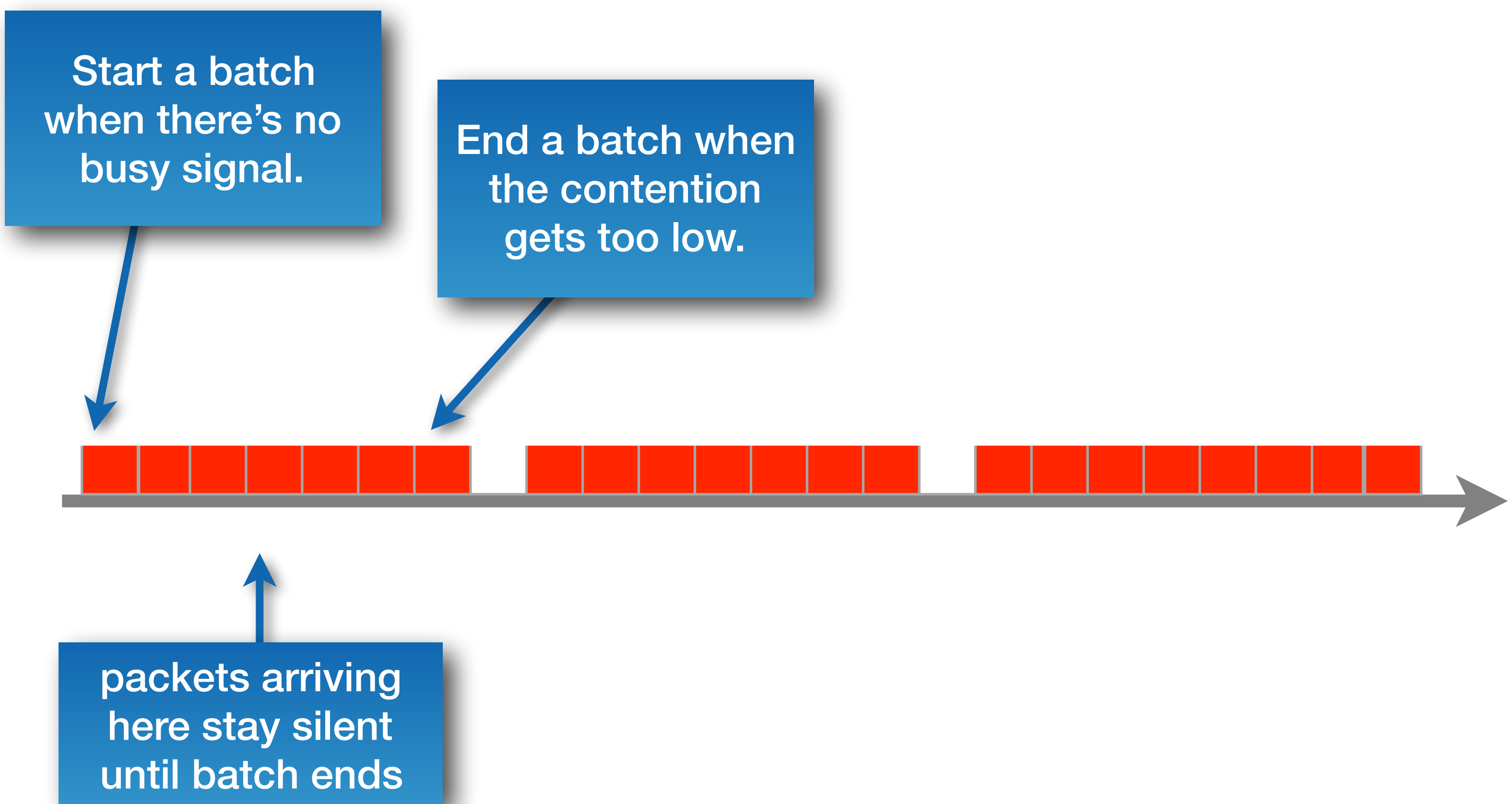
Start a new batch.
(There may still be older batches in the system.)

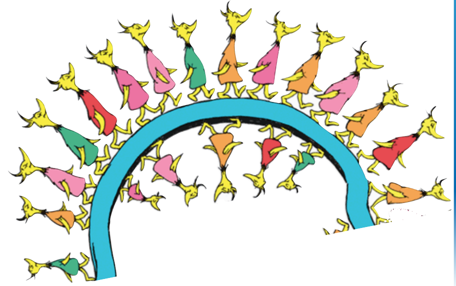
(Not complicated algorithm. Complicated analysis.)



Batches based upon contention

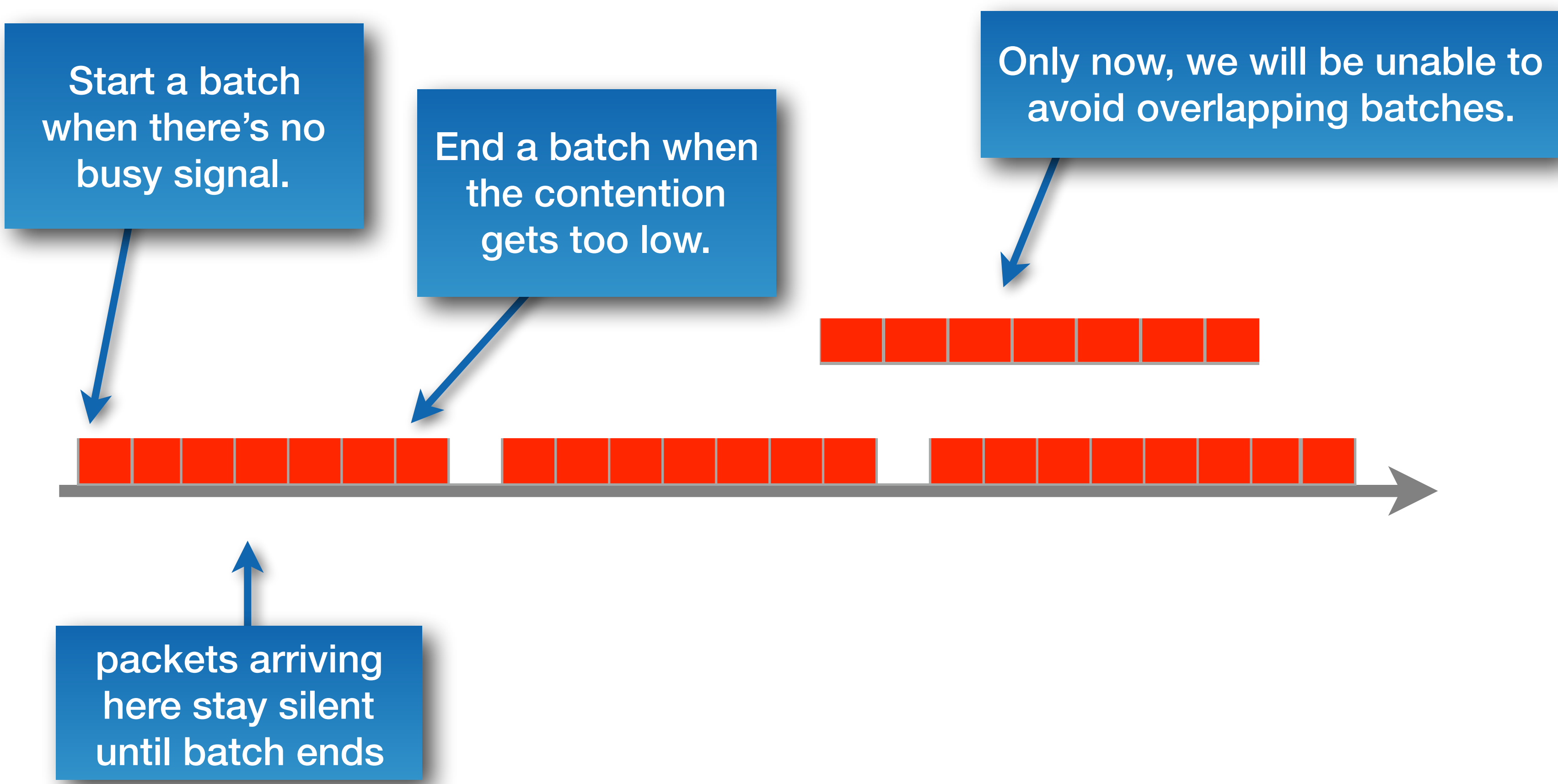
Group packets into synchronized batches.

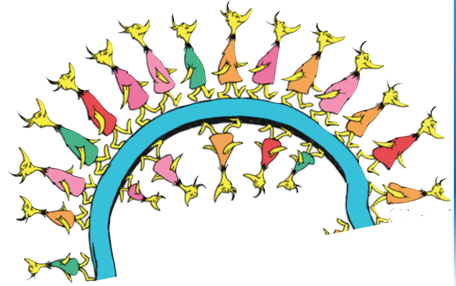




Batches based upon contention

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:

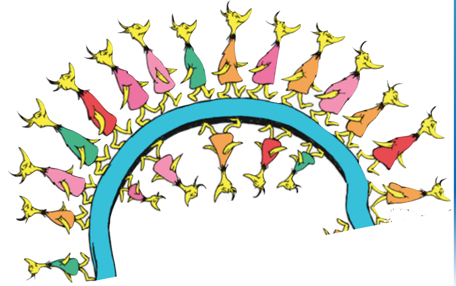
- create a lot of contention,
- but their contention reduces *quickly* as they age.

$1 \rightarrow 1/2 \rightarrow 1/3 \rightarrow 1/4 \rightarrow 1/5 \ldots$

old packets:

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

$1/1000 \rightarrow 1/1001 \rightarrow 1/1002 \rightarrow 1/1003 \rightarrow 1/1004 \ldots$



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.

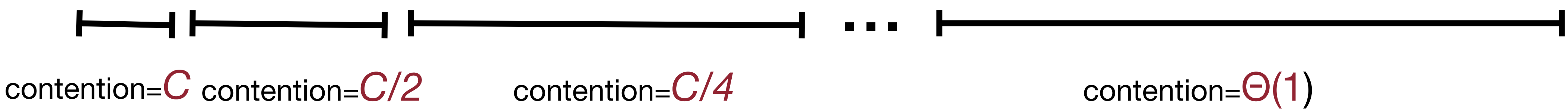
Contention is a slippery parameter.

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

Idea of Structural Argument

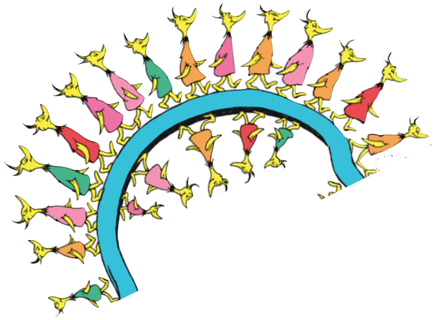
If no new batch joins:

- each time the contention halves, it takes 2X as long before it halves again. (There's no guarantee on how long it takes to halve.)



With constant probability:

- No new batch arrives until the contention is $\Theta(1)$.
- The contention stays $\Theta(1)$ for a long time.
- The contention doesn't shrink to $o(1)$ for too long before a new batch enters the system.



Dynamic Arrivals without Jamming

[Bender, Kopelowitz, Pettie, Young, 15]

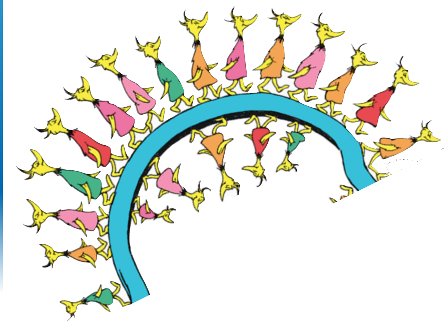
Theorem:

n = # packets.

no jamming of slots.

expected makespan: $O(n)$.

expected # channel accesses: $O(\log \log^* n)$.



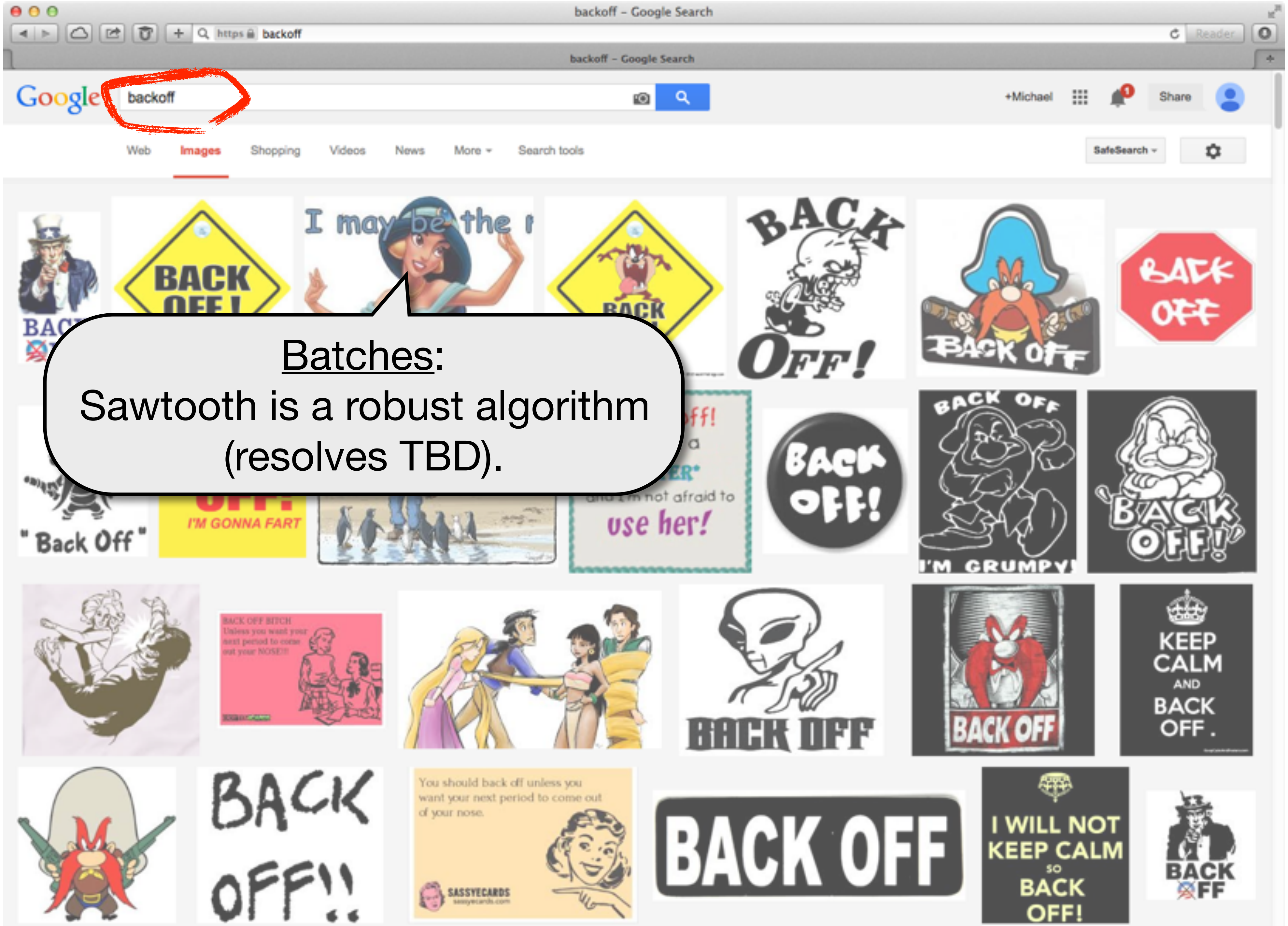
Ideas: Backoff Without Jamming

Active packets collectively estimate *n*.

Then they run sawtooth with the right Θ(*n*).

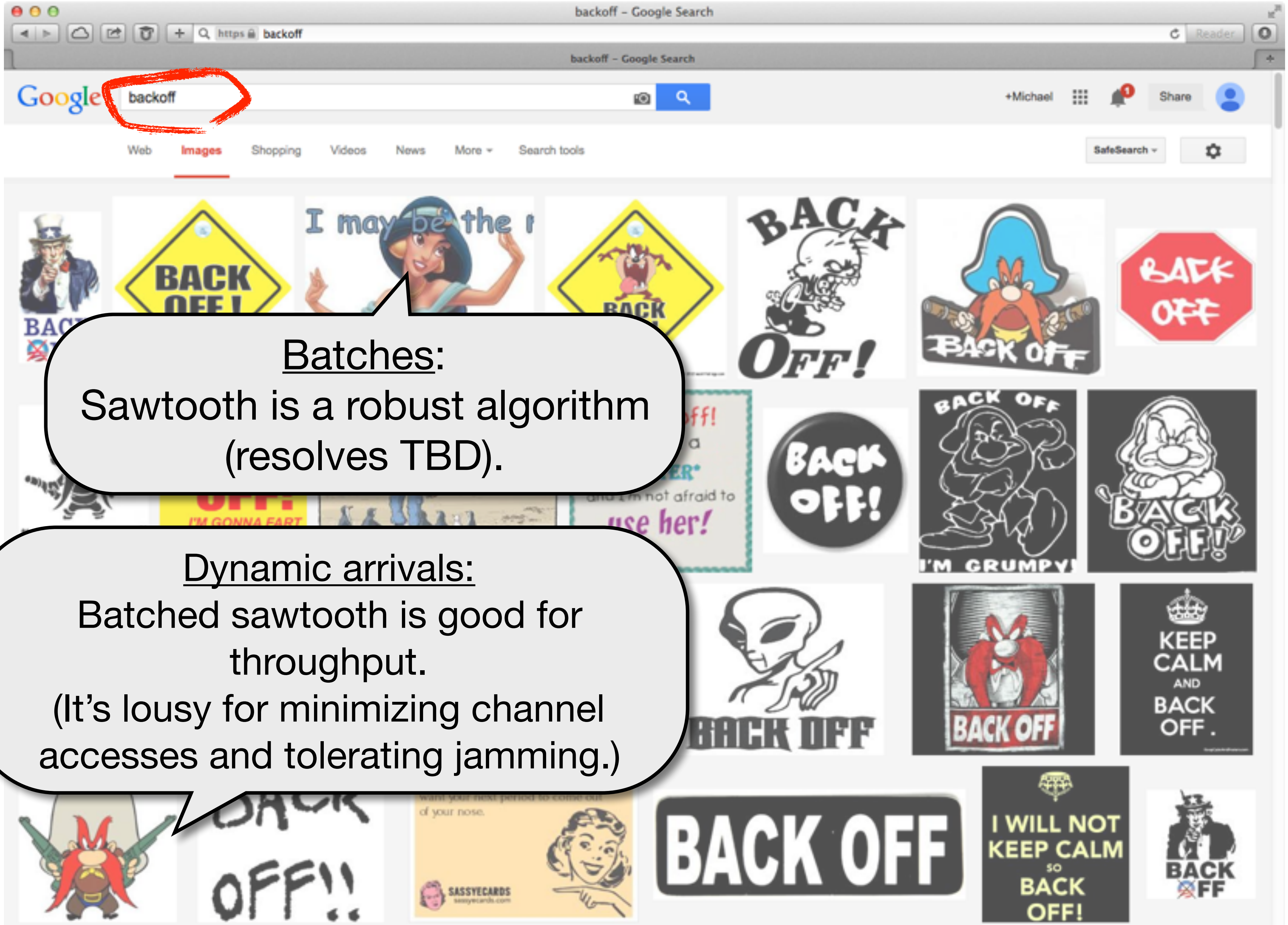
The hardest part of estimating *n*, is estimating log*n*.**

Morals for better backoff algorithms



<u>Batches</u>:
Sawtooth is a robust algorithm
(resolves TBD).

Morals for better backoff algorithms



<u>Batches</u>:
Sawtooth is a robust algorithm (resolves TBD).

<u>Dynamic arrivals:</u>
Batched sawtooth is good for throughput.
(It's lousy for minimizing channel accesses and tolerating jamming.)

Morals for better backoff algorithms

<u>Batches</u>:
Sawtooth is a robust algorithm (resolves TBD).

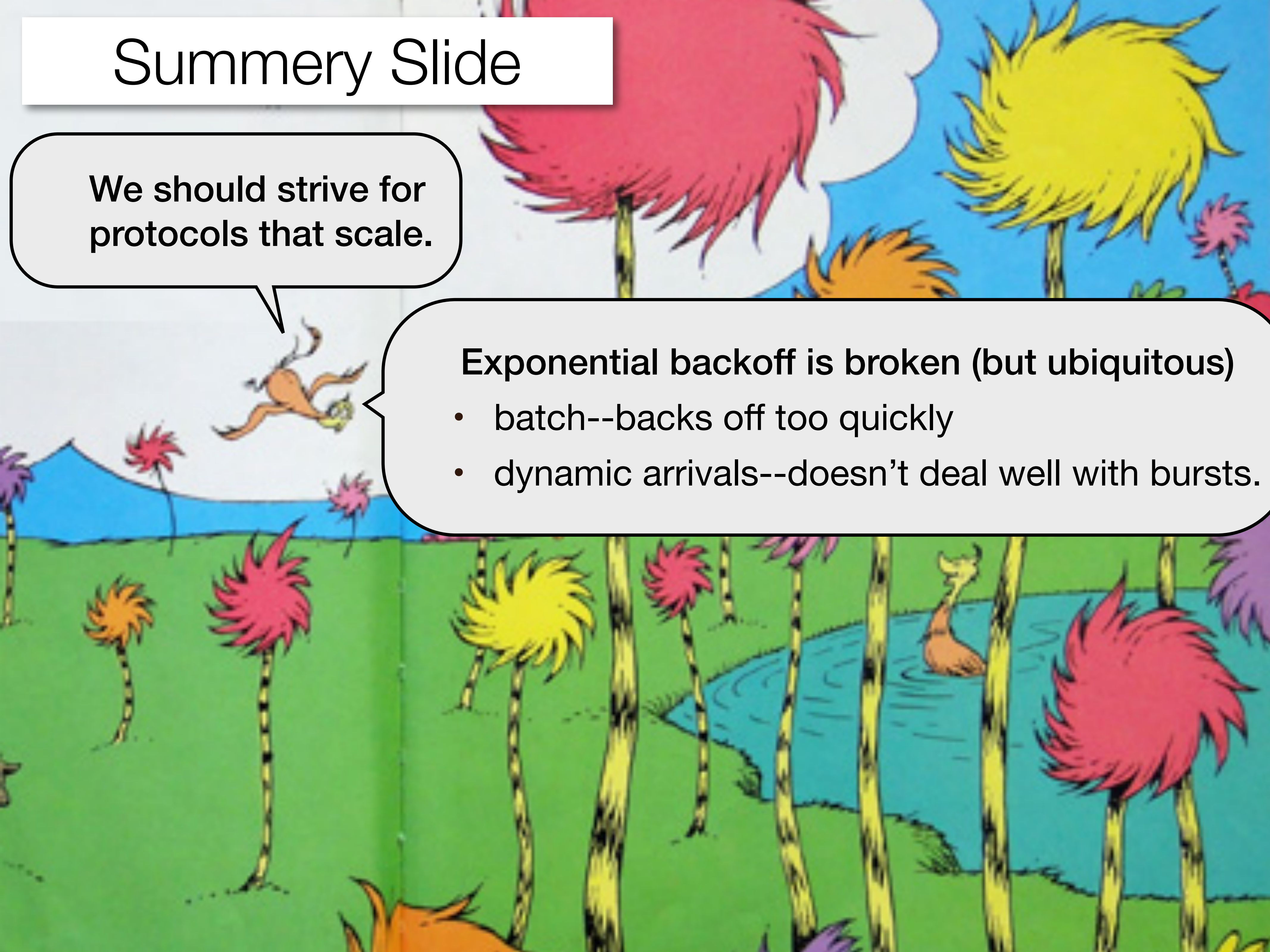
<u>Dynamic arrivals:</u>
Batched sawtooth is good for throughput.
(It's lousy for minimizing channel accesses and tolerating jamming.)

<u>Dynamic arrivals:</u>
Two scalable backoff protocols that resolve TBDs.

Summery Slide

Summery Slide

Summery Slide



Summery Slide

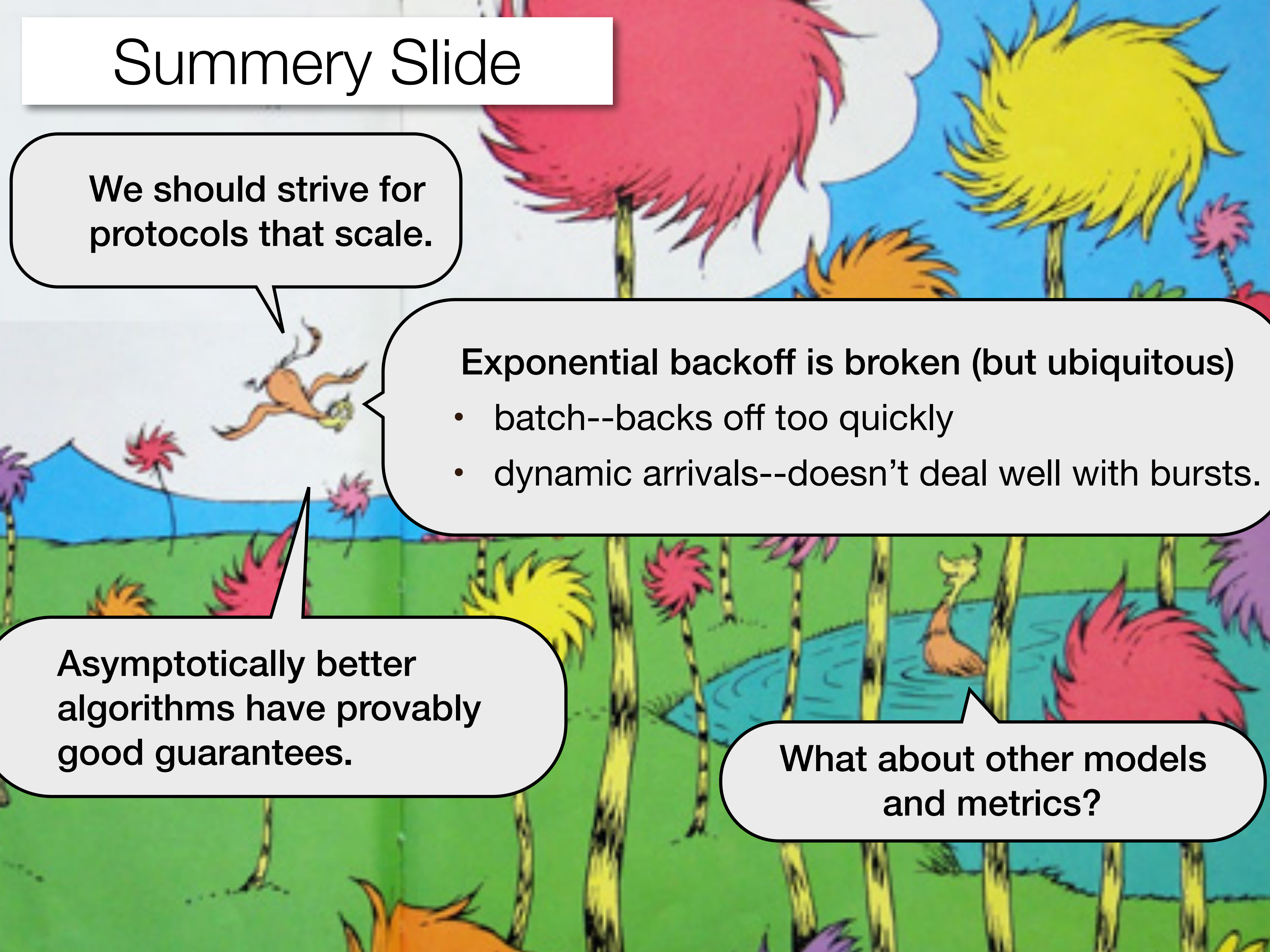
We should strive for protocols that scale.

Exponential backoff is broken (but ubiquitous)

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

Asymptotically better algorithms have provably good guarantees.

Summery Slide



We should strive for protocols that scale.

Exponential backoff is broken (but ubiquitous)

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

Asymptotically better algorithms have provably good guarantees.

What about other models and metrics?