Parallel DBs

April 25, 2017

Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>

Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Send me rows with a 'B' in the bloom filter summarizing the set {2,3,6}

Node 2

<2,X>
<3,Y>
<6,Y>

Sending Hints

$R_k \bowtie_B S_i$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

<2,B>
<2,C>
<3,D>
<4,E>

Send me rows with a 'B' in the bloom filter summarizing the set {2,3,6}

Node 2

<2,X>
<3,Y>
<6,Y>

This is called a <u>bloom-join</u>.

Bloom Filter Construction

Empty Filter (Size: m = 20)

00000000000000000000

Use hash functions to pick a fixed number of bits (k = 3)

$h_1(X) = 13$; $h_2(X) = 2$; $h_3(X) = 5$

Set those bits to 1

00100100000001000000

Bloom Filter Lookup

Key 1	00101010
Key 2	01000110
Key 3	10000110
Key 4	01001100

Filters are combined by Bitwise-OR

e.g. (Key 1 | Key 2)

= 01101110

How do we test for inclusion?

(Key & Filter) == Key?

(Key 1 & S) = 00101010 ✓

(Key 3 & S) = 00000110 ✗

(Key 4 & S) = 01001100 ✓

False Positive

Bloom Filter Parameters

m = size of the bit vector

Bigger – More space used
Smaller – More false positives

k = # of bits set per element

More Bits – More false positives
Fewer Bits – More false positives
(Need to balance #)

Bloom Filters

How do we pick M and K?

Bloom Filters

Probability that 1 bit is set by 1 hash fn

$1/m$

Bloom Filters

Probability that 1 bit is not set by 1 hash fn

$$1 - \ 1/m$$

Bloom Filters

Probability that 1 bit is not set by k hash fns

$$(1 - 1/m)^k$$

Bloom Filters

Probability that 1 bit is not set by k hash fns
for n records

$$(1 - 1/m)^{k n}$$

So for an arbitrary record, what is the probability
that all of its bits will be set?

Bloom Filters

Probability that 1 bit is set by k hash fns
for n records

$$1 - (1 - 1/m)^{k n}$$

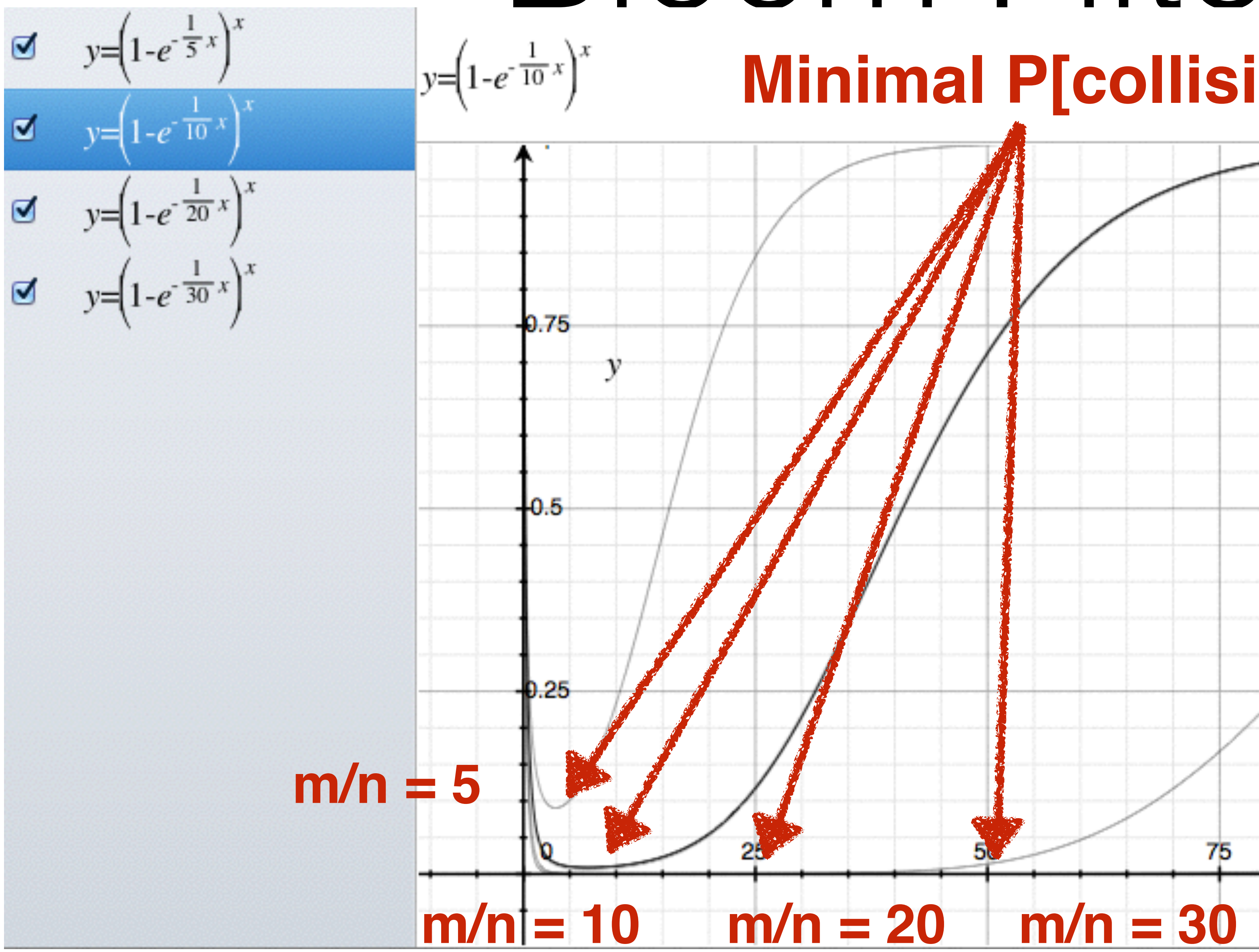
Bloom Filters

Probability that all k bits are set by k hash fns
for n records

$$\approx \left(1 - \left(1 - 1/m\right)^{kn}\right)^{k}$$

$$\approx \left(1 - e^{-kn/m}\right)^{k}$$

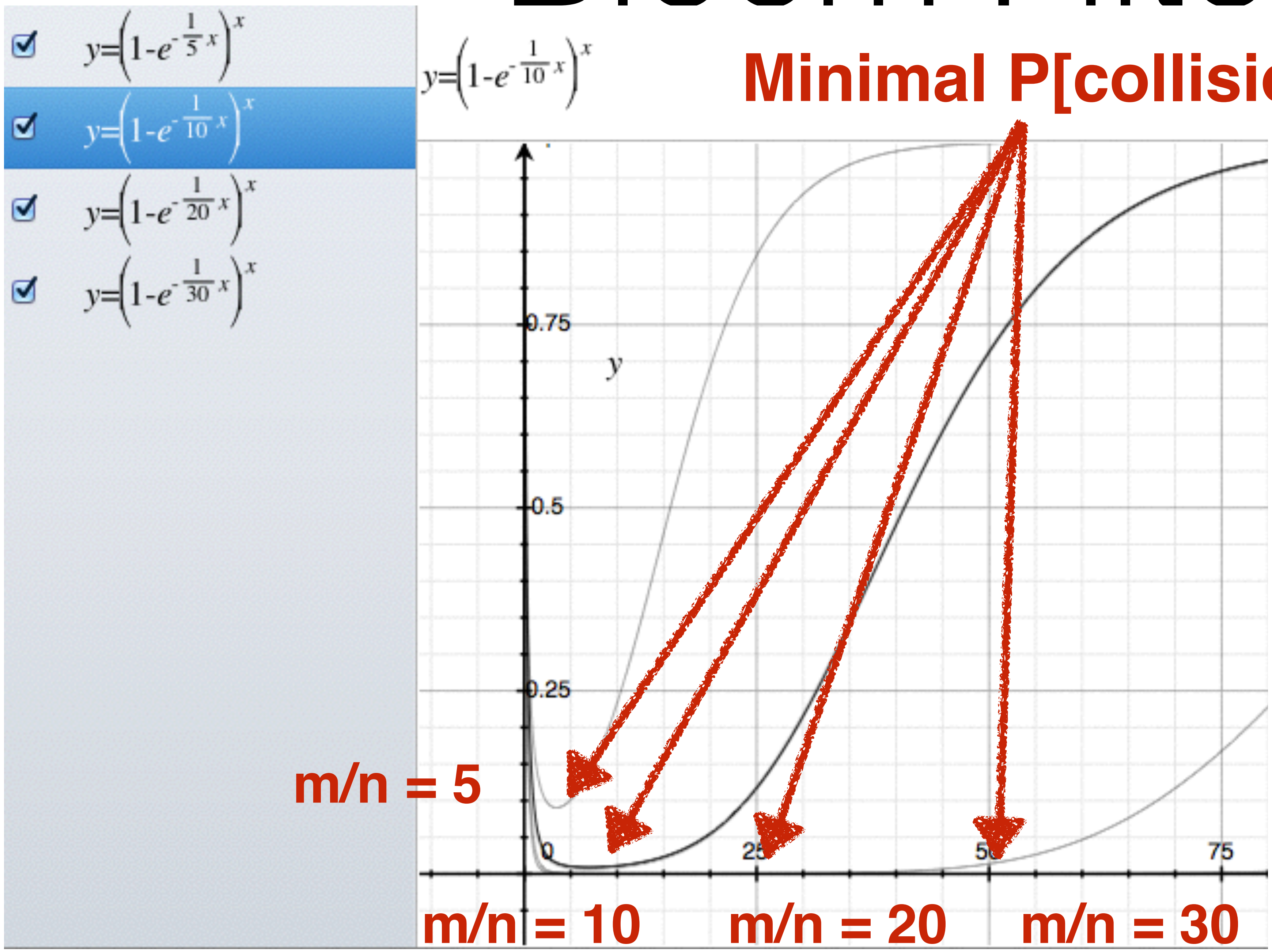
Bloom Filters



$^m/_n$	k	p(collision)
5	3	~9.2%
10	8	~0.85%
20	14	~0.007%
30	21	~0.000055%

Minimal P[collision] is at $k \approx 0.7 \cdot m/n$

Bloom Filters



$^m/_n$	k	p(collision)
5	3	~9.2%
10	8	~0.85%
20	14	~0.007%
30	21	~0.000055%

5 bits/record, 3 bits set = 10% chance of collision

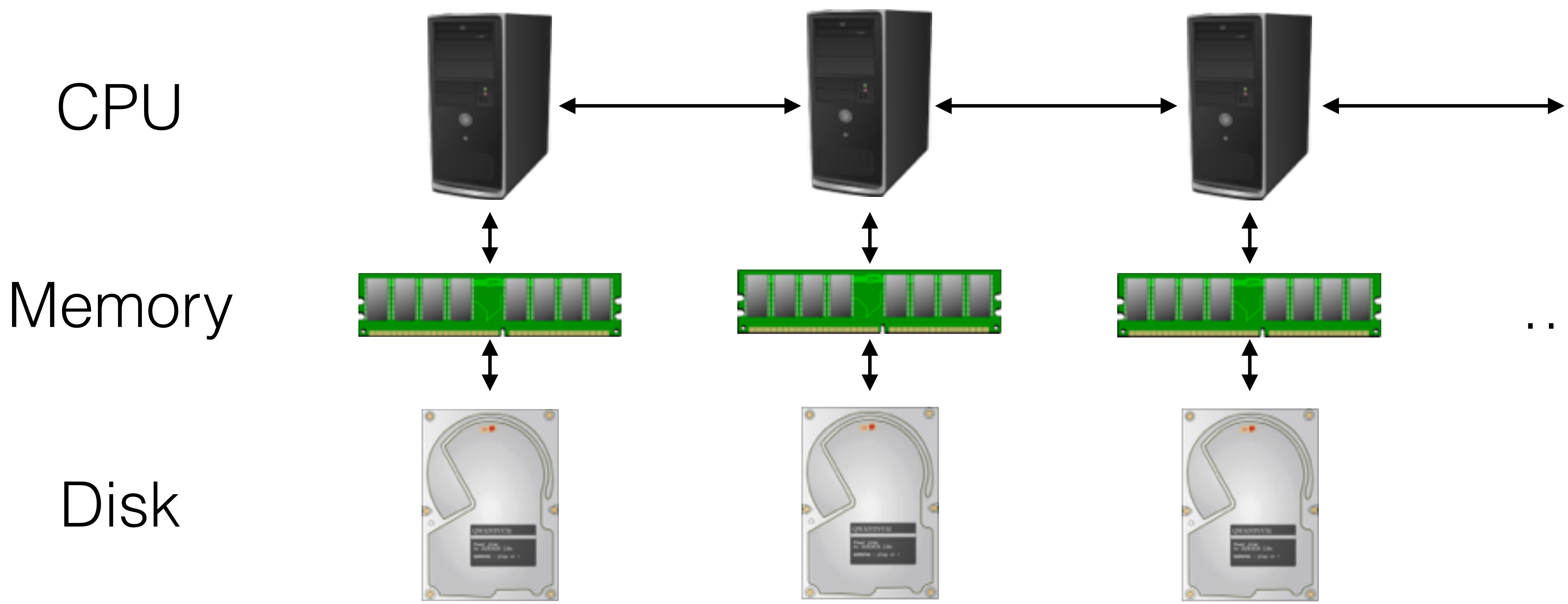
Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates

Parallelism Models

Option 4: "Shared Nothing" in which all communication is explicit.



We'll be talking about "shared nothing" for updates.
Other models are easier to work with.

Data Parallelism

Replication

(needed for safety)

Partitioning

Updates

What can go wrong?

- Non-Serializable Schedules

Node 1

```
T1: W(X)
T2: W(X)
T2: W(Y)
T1: W(Y)
```

Updates (in Parallel)

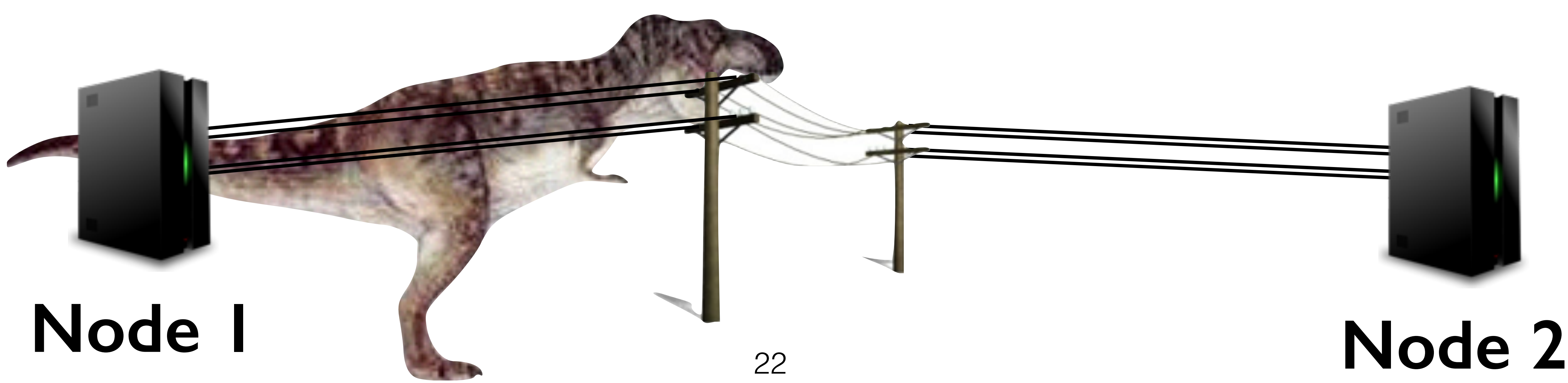
What can go wrong?

- Non-Serializable Schedules

Updates (in Parallel)

What can go wrong?

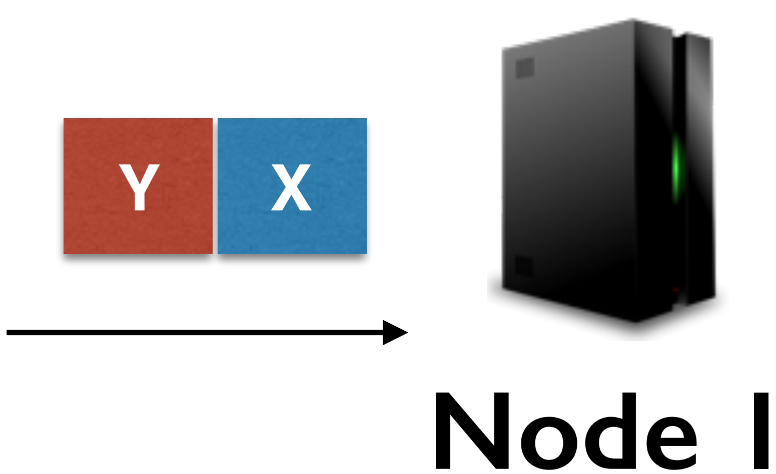
- Non-Serializable Schedules
- One Compute Node Fails

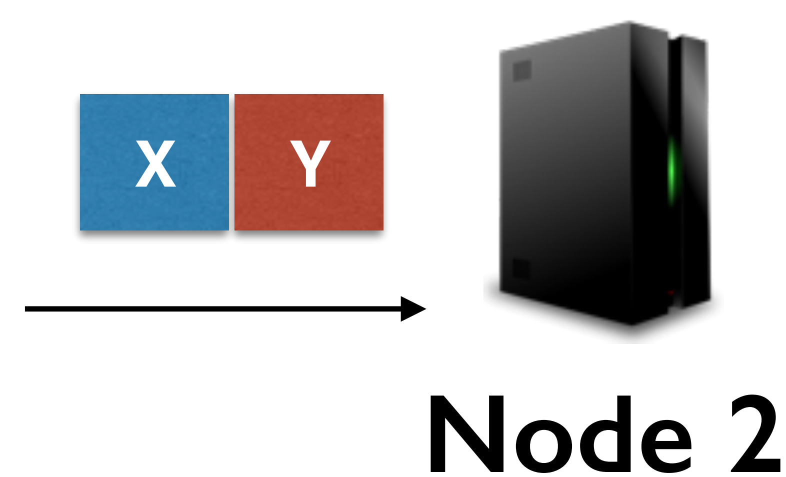


Updates (in Parallel)

What can go wrong?

- Non-Serializable Schedules
- One Compute Node Fails
- A Communication Channel Fails
- Messages delivered out-of-order





Updates (in Parallel)

What can go wrong?

- Non-Serializable Schedules → Classical Xacts
- One Compute Node Fails → "Partitions"
- A Communication Channel Fails → "Partitions"
- Messages delivered out-of-order → Consensus

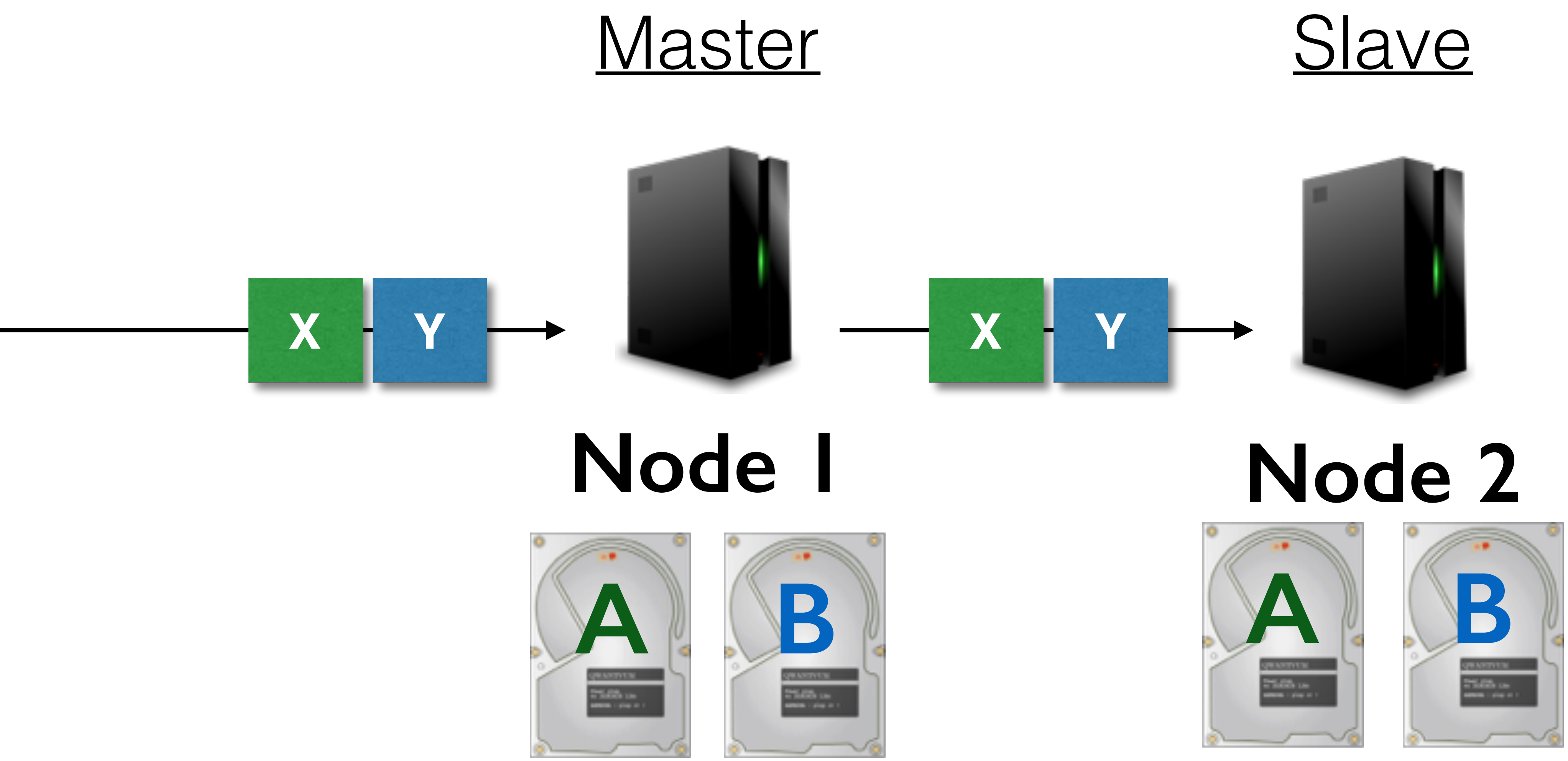
Data Parallelism

Replication

Partitioning

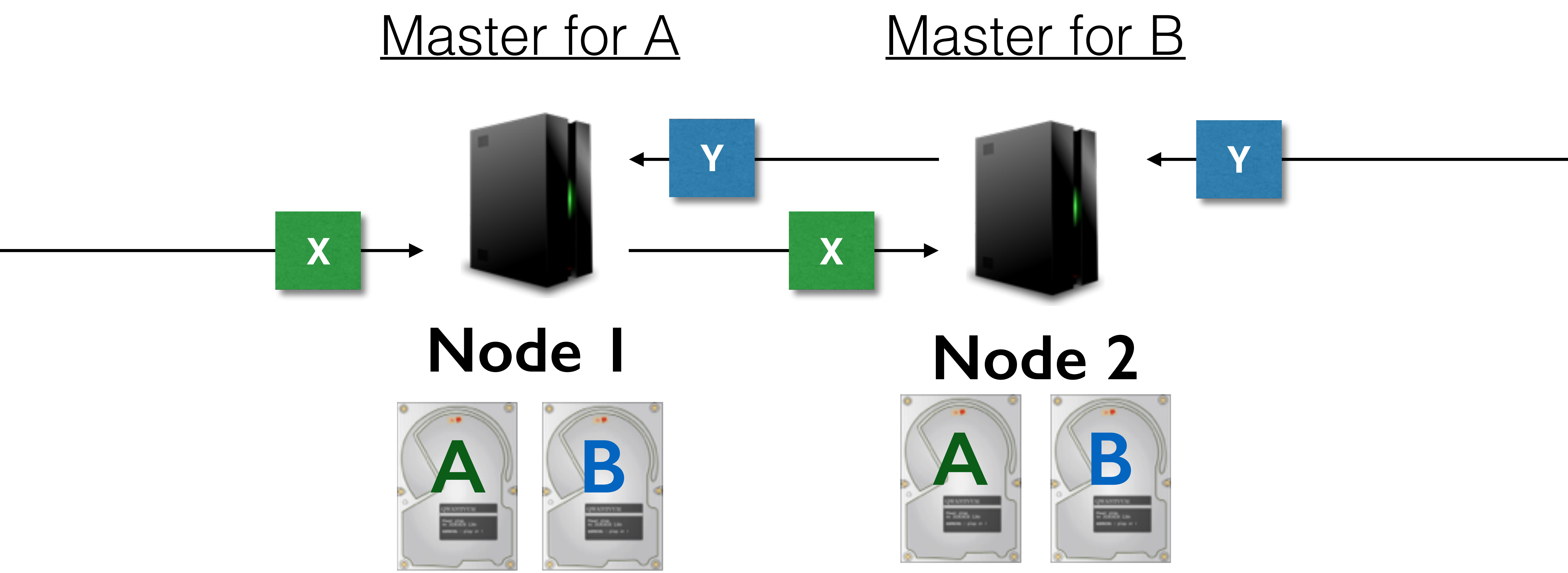
(needed for safety)

Simple Consensus



“Safe” … but Node 1 is a bottleneck.

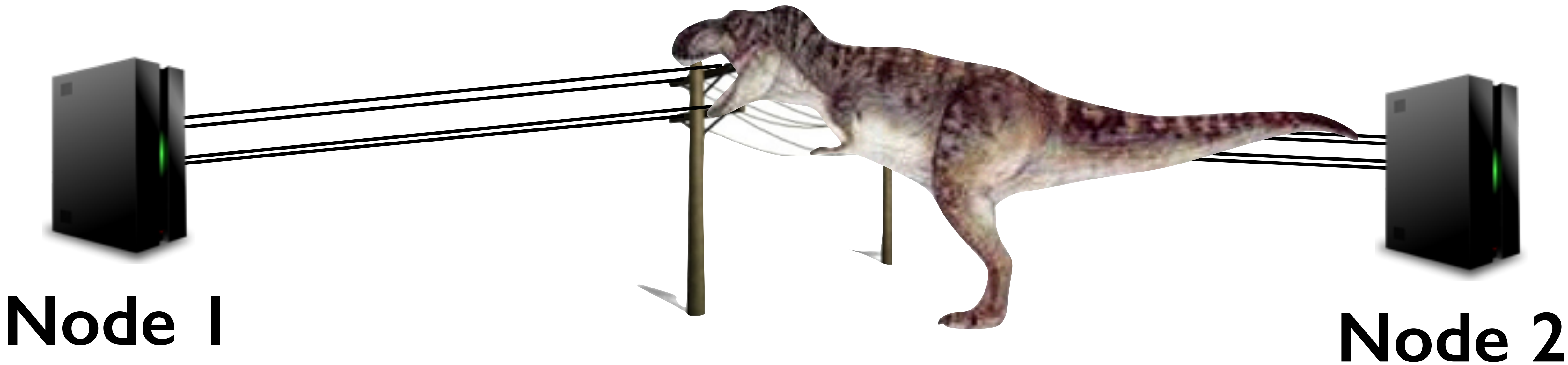
Simpl-ish Consensus



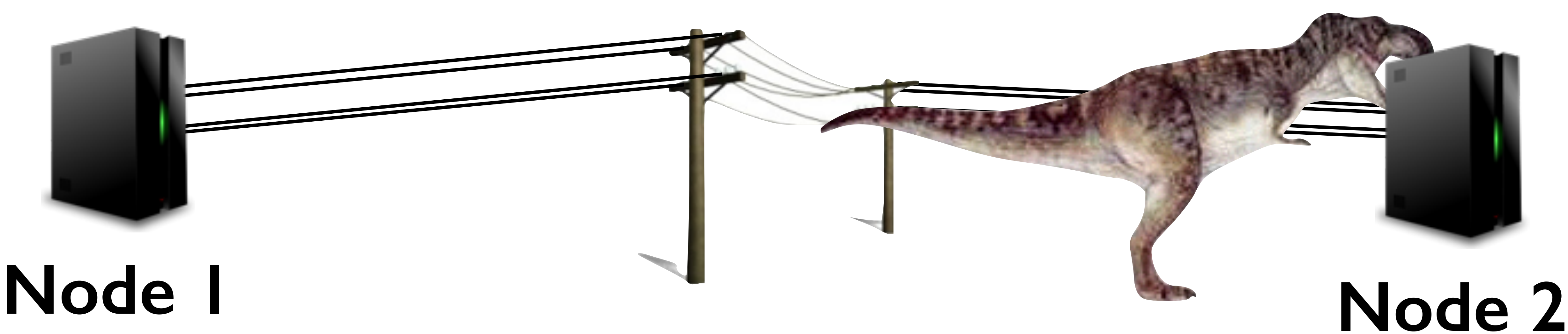
Node 2 agrees to Node 1's order for A.
Node 1 agrees to Node 2's order for B.

Partitions

Channel Failure



Node Failure



From Node 1's perspective, how are these cases different?

They’re not!

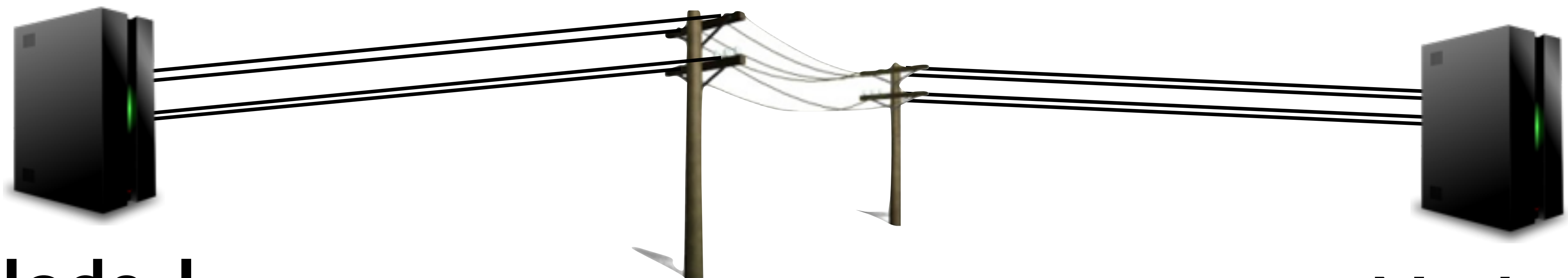
Failure Recovery

- Node Failure
  - The node restarts and resumes serving requests.
- Channel Failure
  - Node 1 and Node 2 regain connectivity.

Partitions

A=1
B=5

A=1
B=5



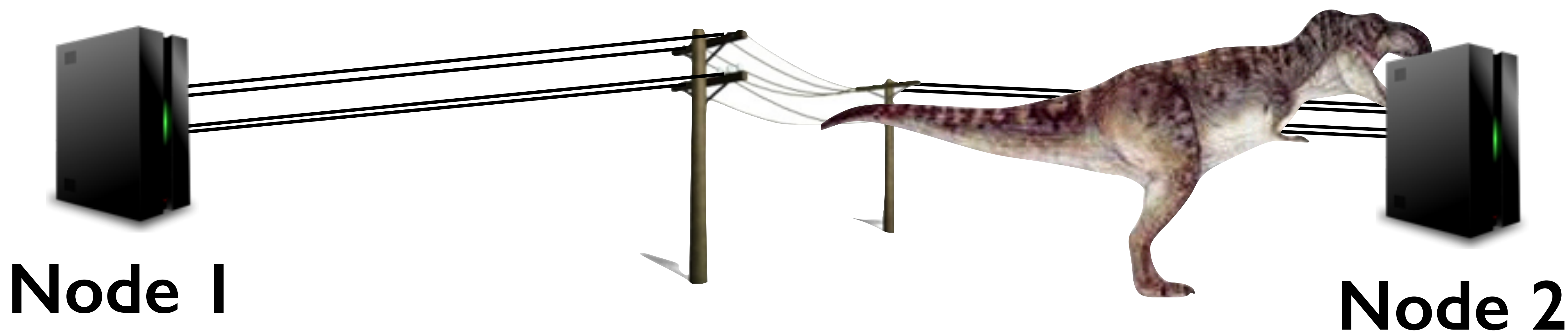
Node 1

Node 2

Partitions

A=1
B=5

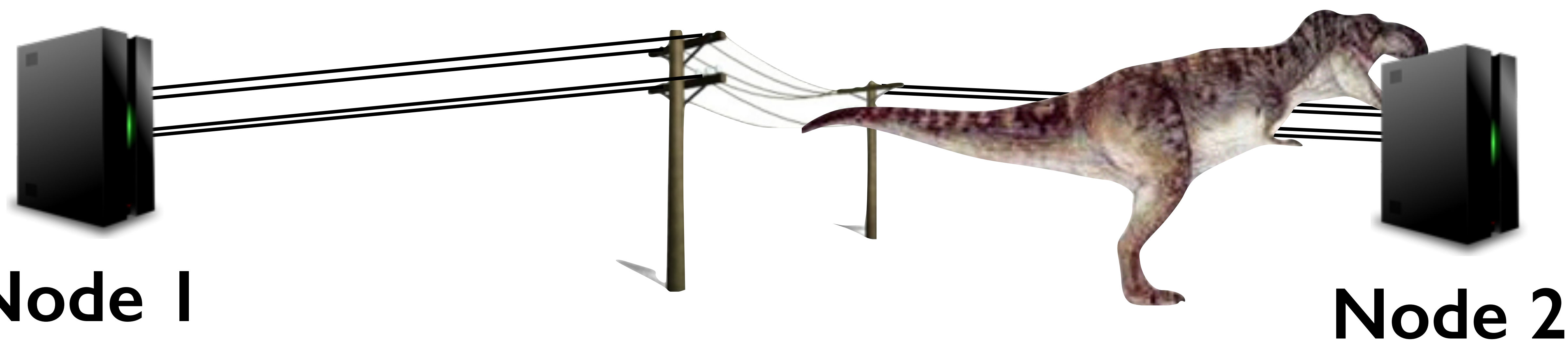
Option 1: Node 1 takes over



Partitions

Option 1: Node 1 takes over

A=1
B=5

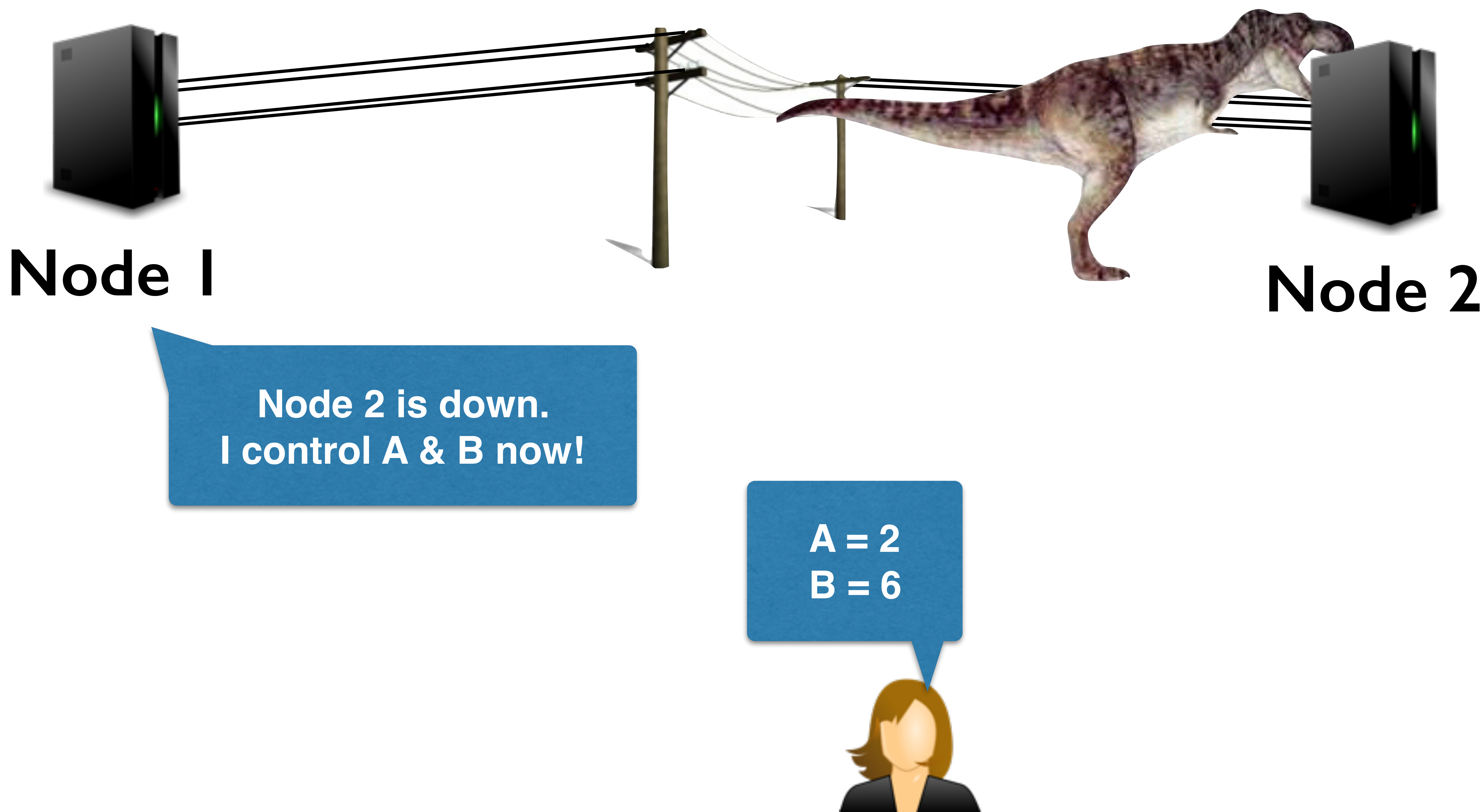


Node 2 is down.
I control A & B now!

Partitions

Option 1: Node 1 takes over

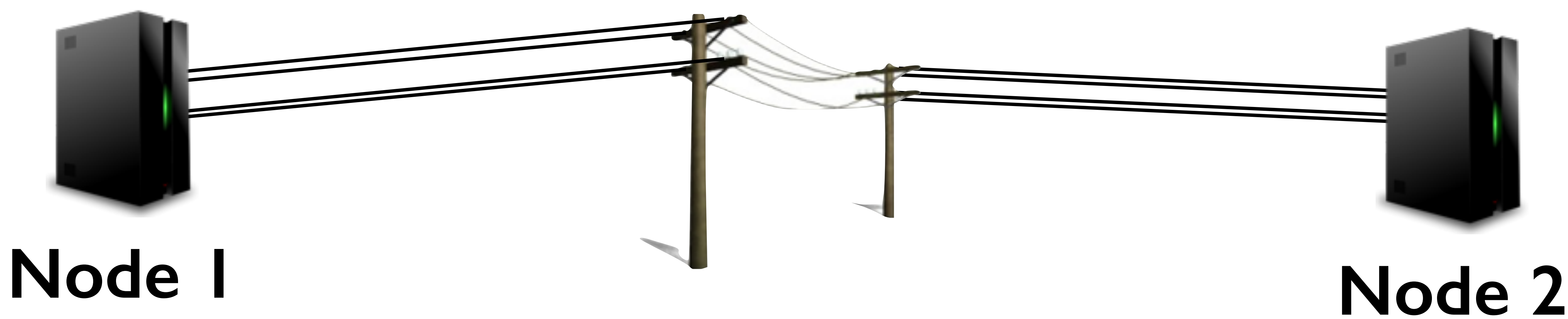
A=2
B=6



Partitions

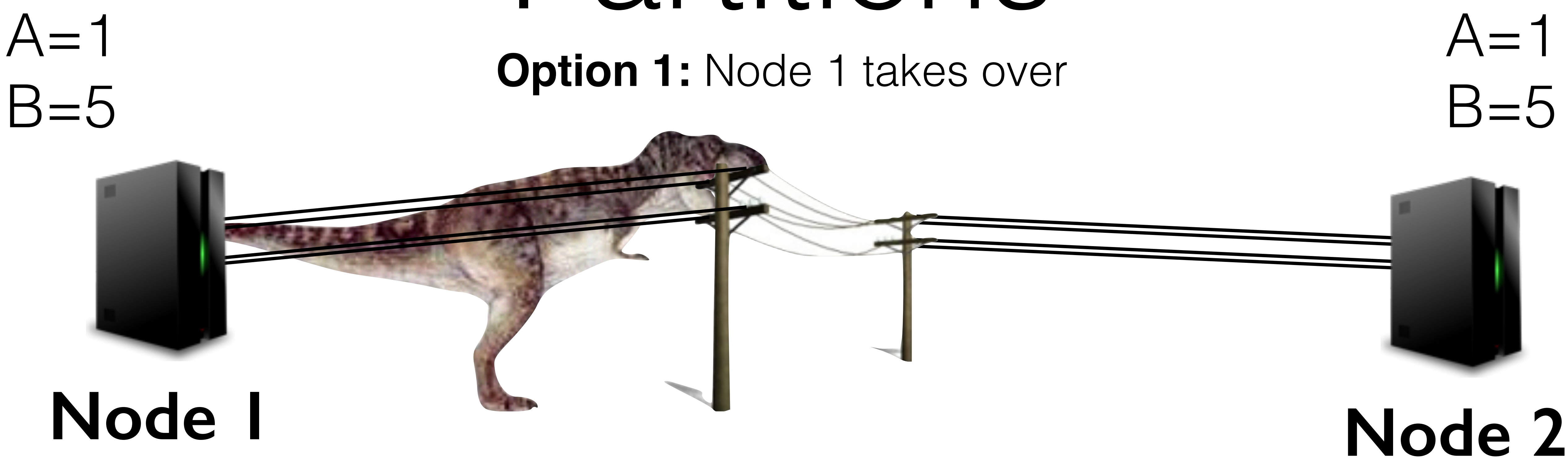
A=2
B=6

Option 1: Node 1 takes over



Partitions

Option 1: Node 1 takes over



Partitions

Option 1: Node 1 takes over

A=1
B=5

A=1
B=5

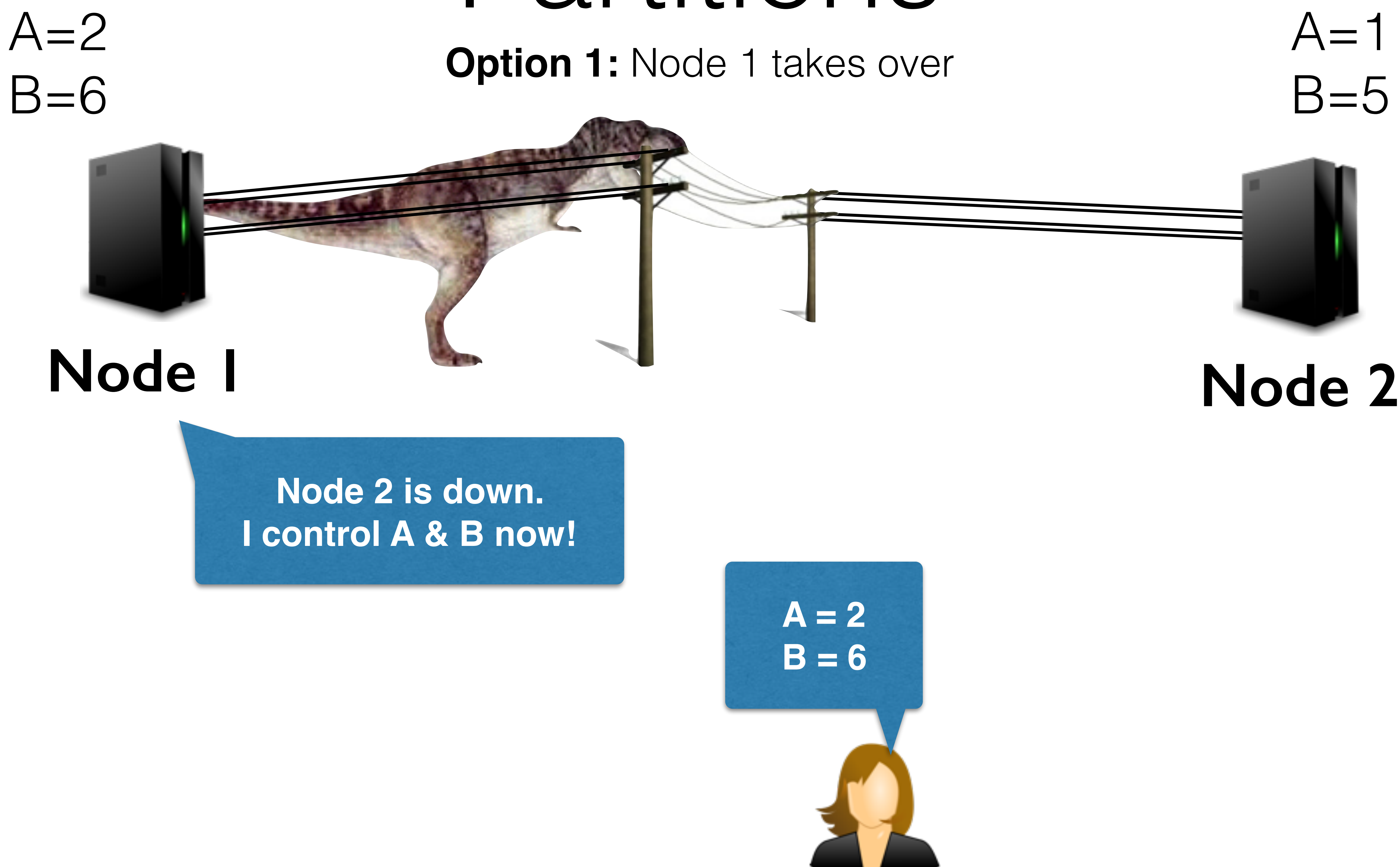
Node 1

Node 2

Node 2 is down.
I control A & B now!

Partitions

Option 1: Node 1 takes over

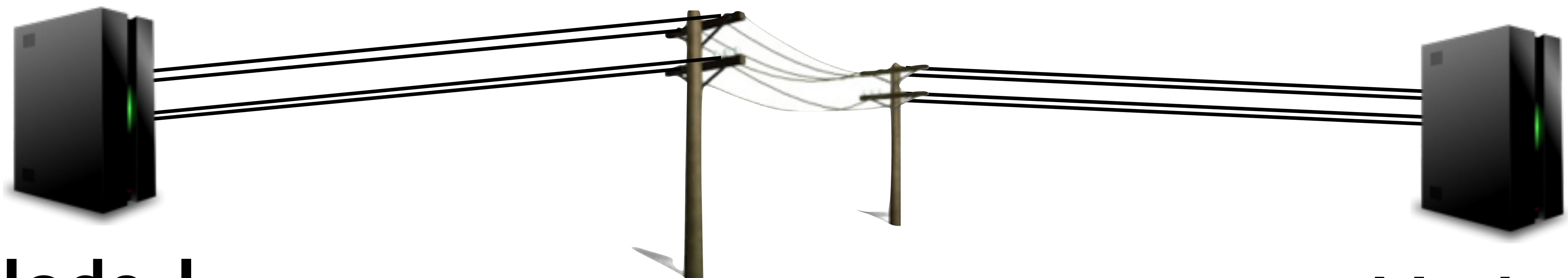


Partitions

A=2
B=6

Option 1: Node 1 takes over

A=1
B=5



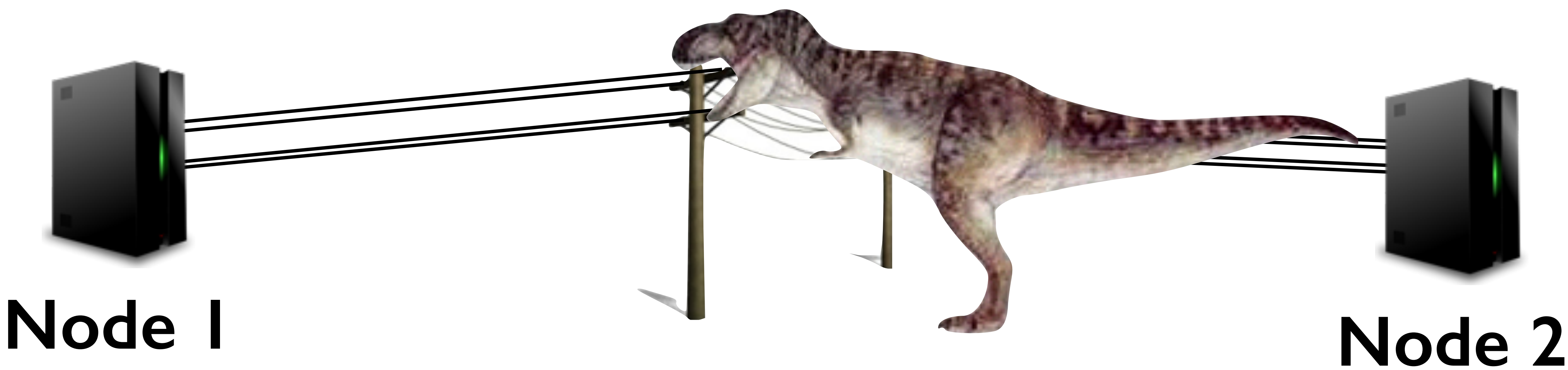
Node 1

Node 2

INCONSISTENCY!

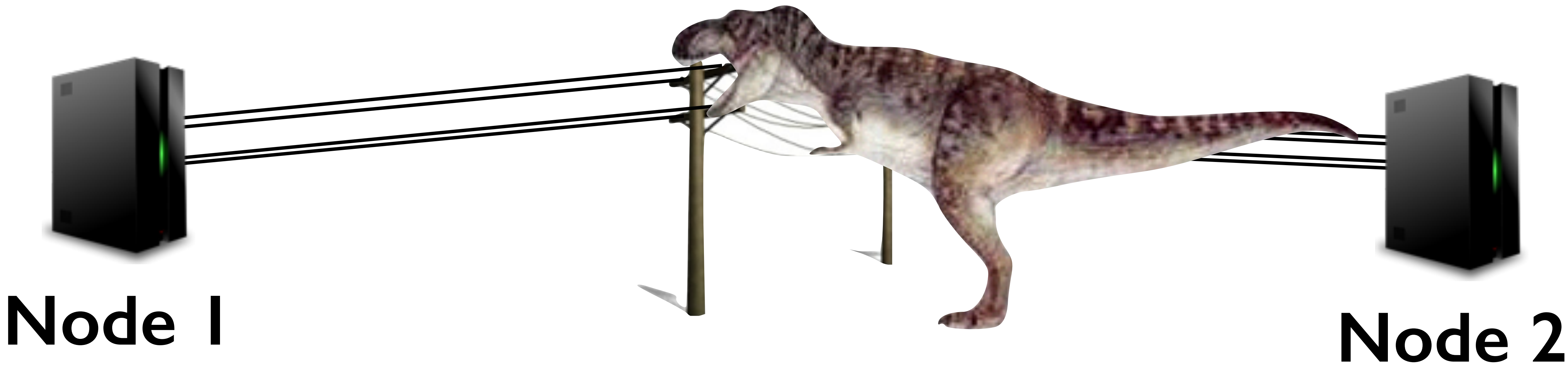
Partitions

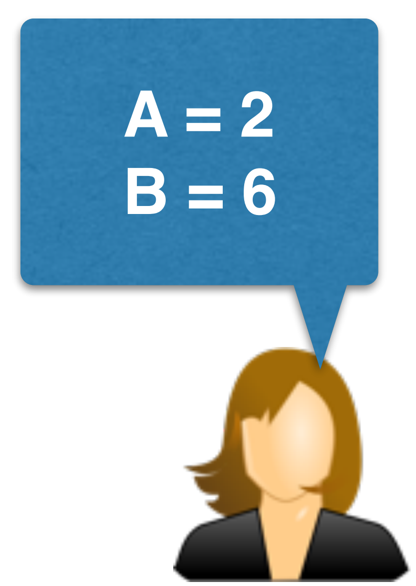
Option 2: Wait



Partitions

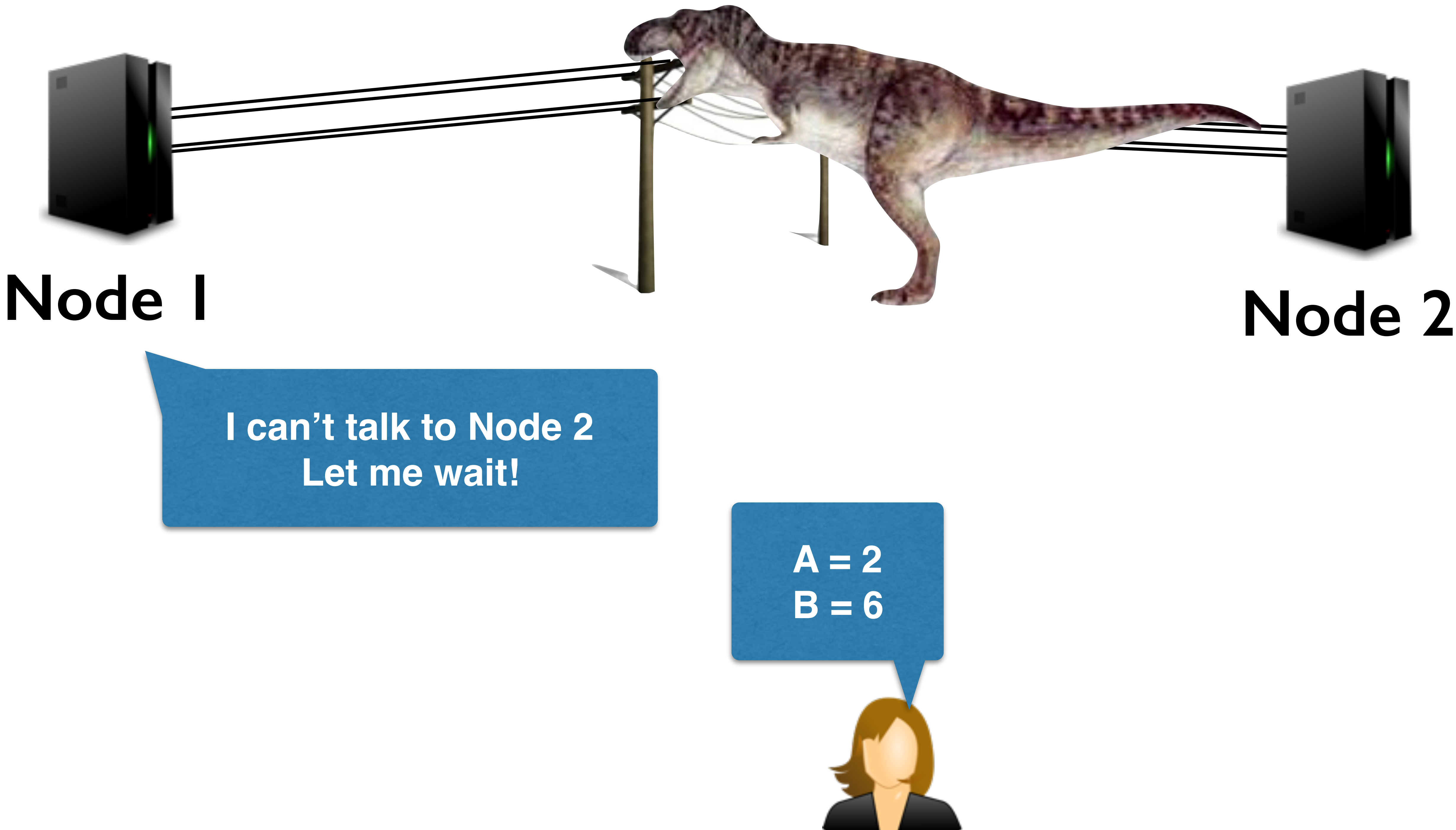
Option 2: Wait





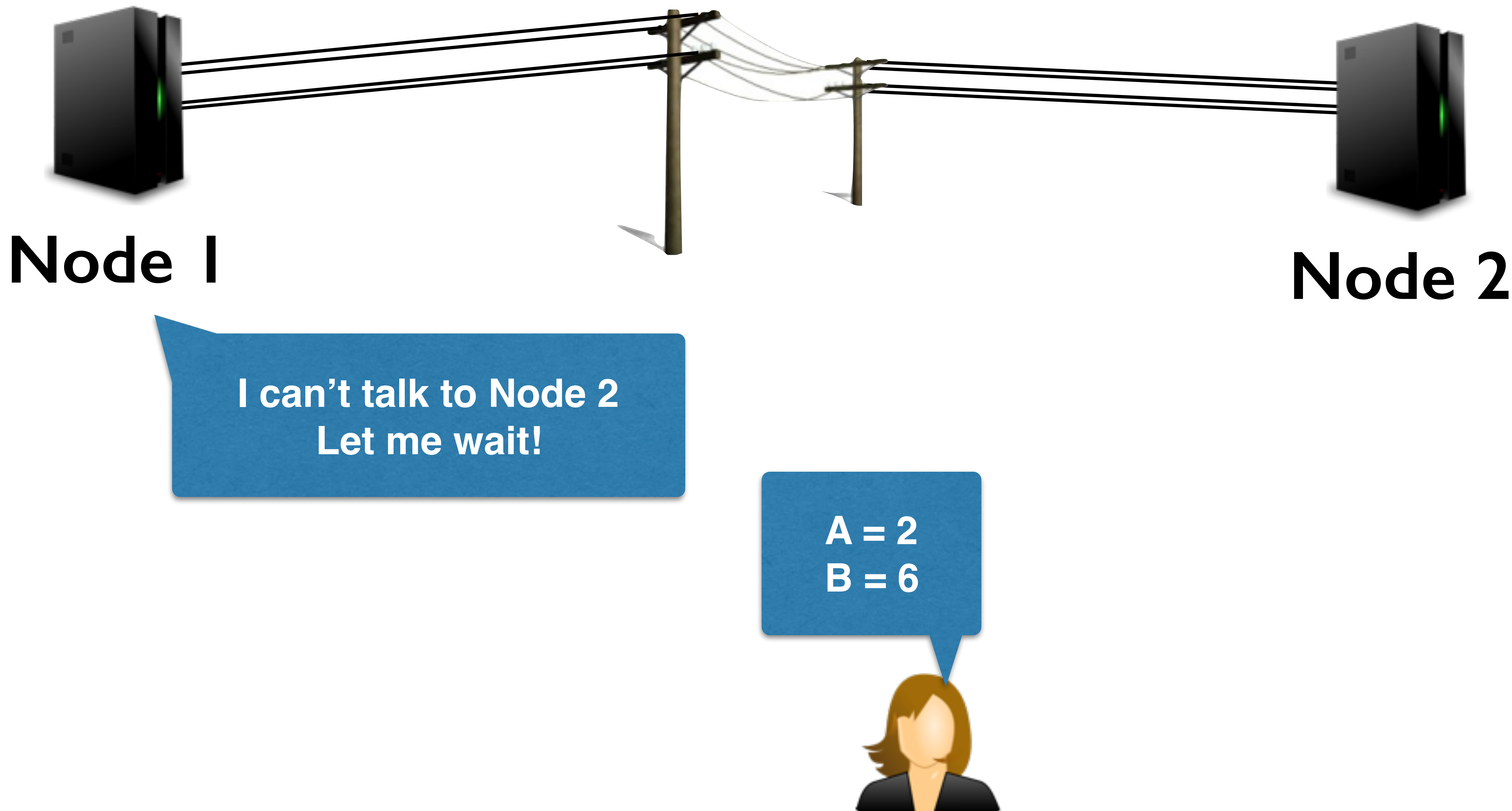
Partitions

Option 2: Wait



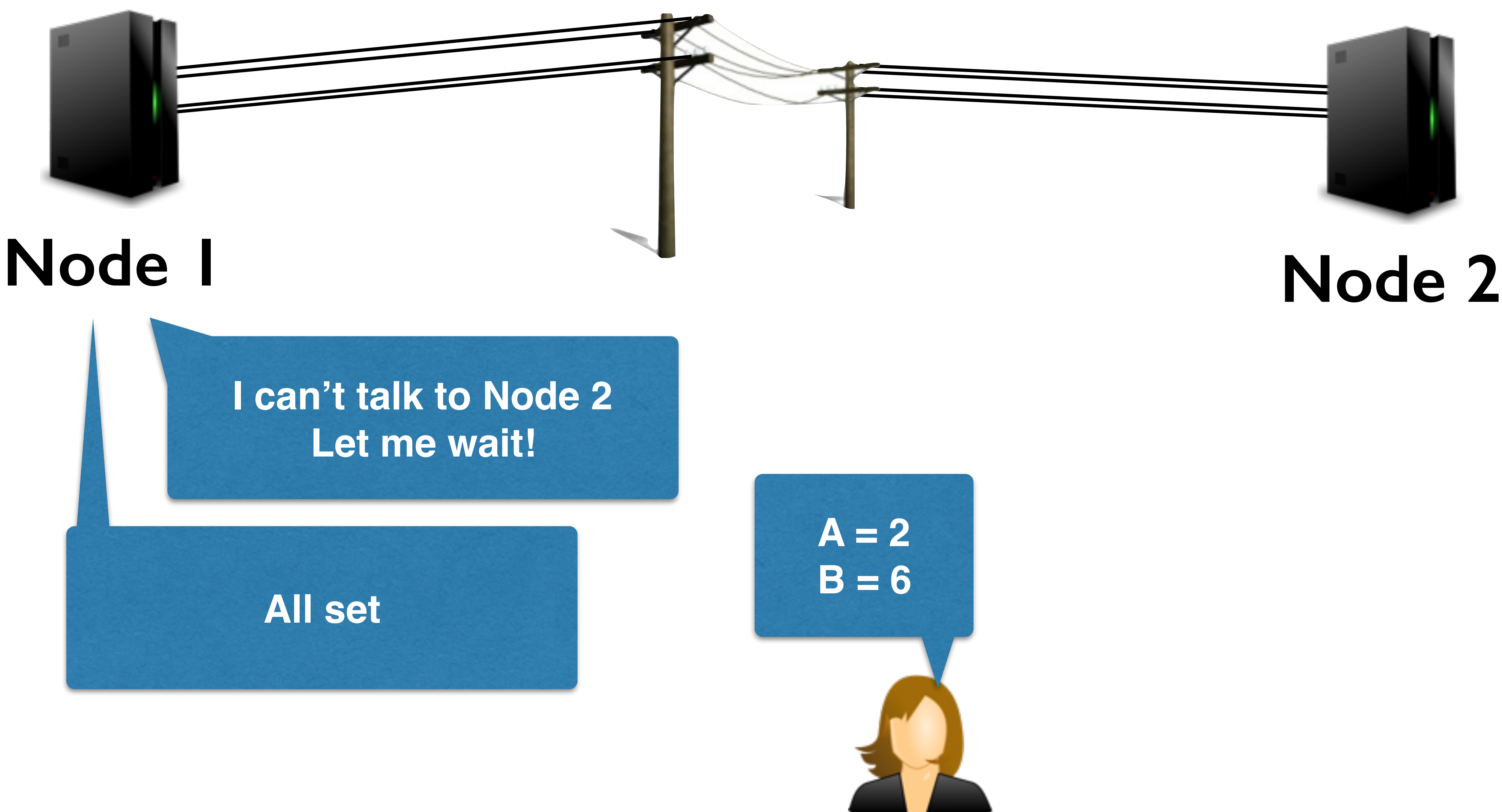
Partitions

Option 2: Wait



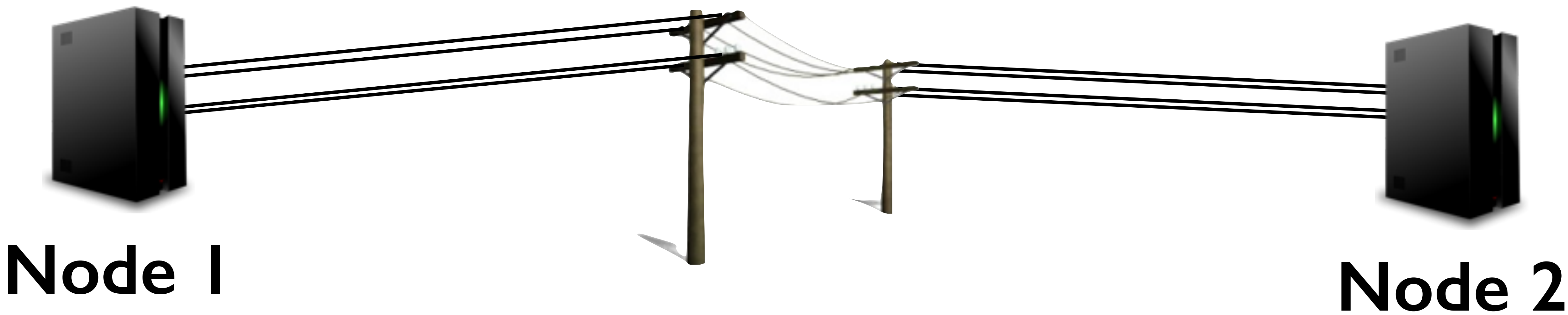
Partitions

Option 2: Wait



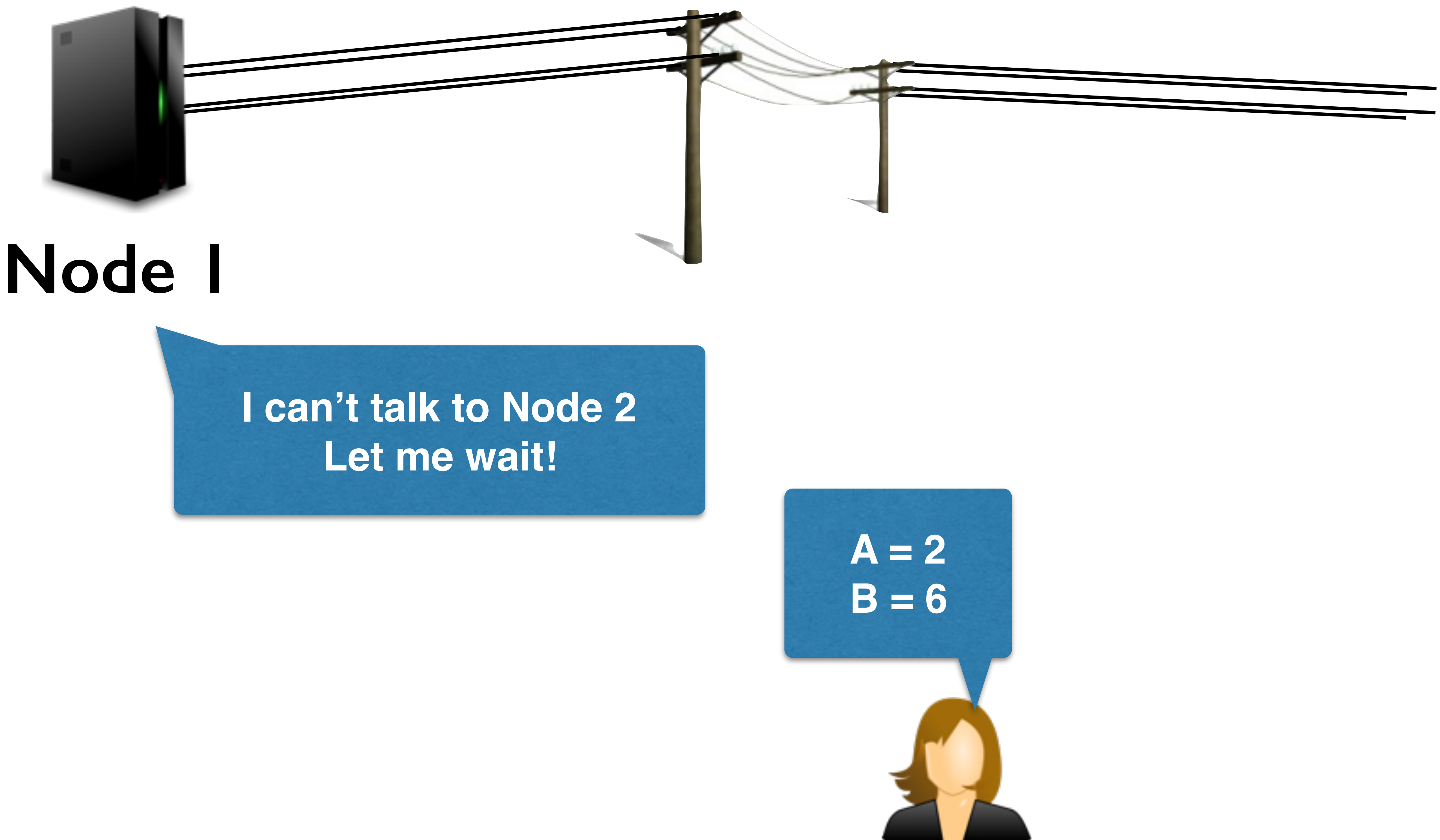
Partitions

Option 2: Wait



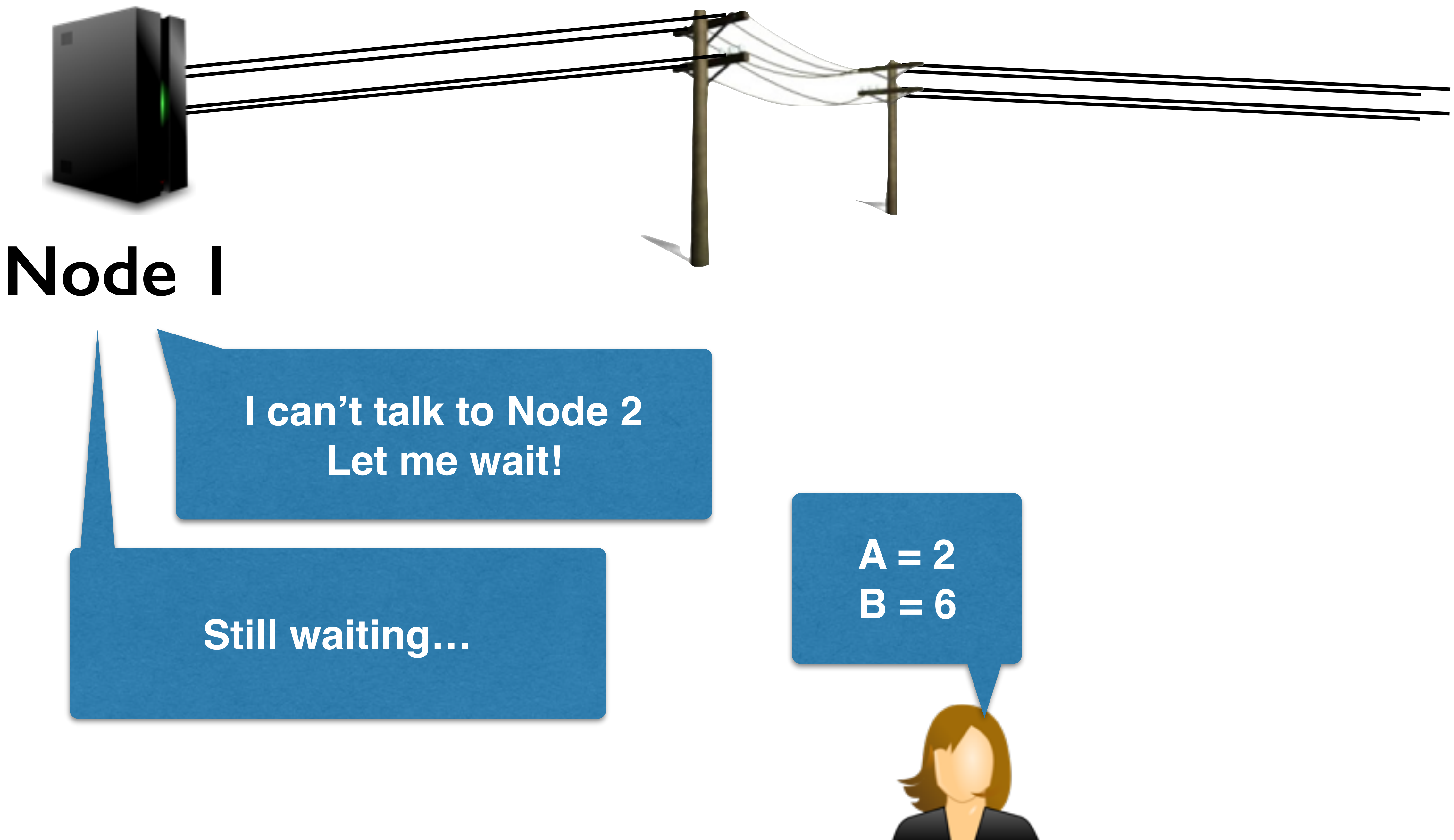
Partitions

Option 2: Wait



Partitions

Option 2: Wait



Partitions

Option 1: Assume Node Failure

All data is available… but at risk of inconsistency.

Option 2: Assume Connection Failure

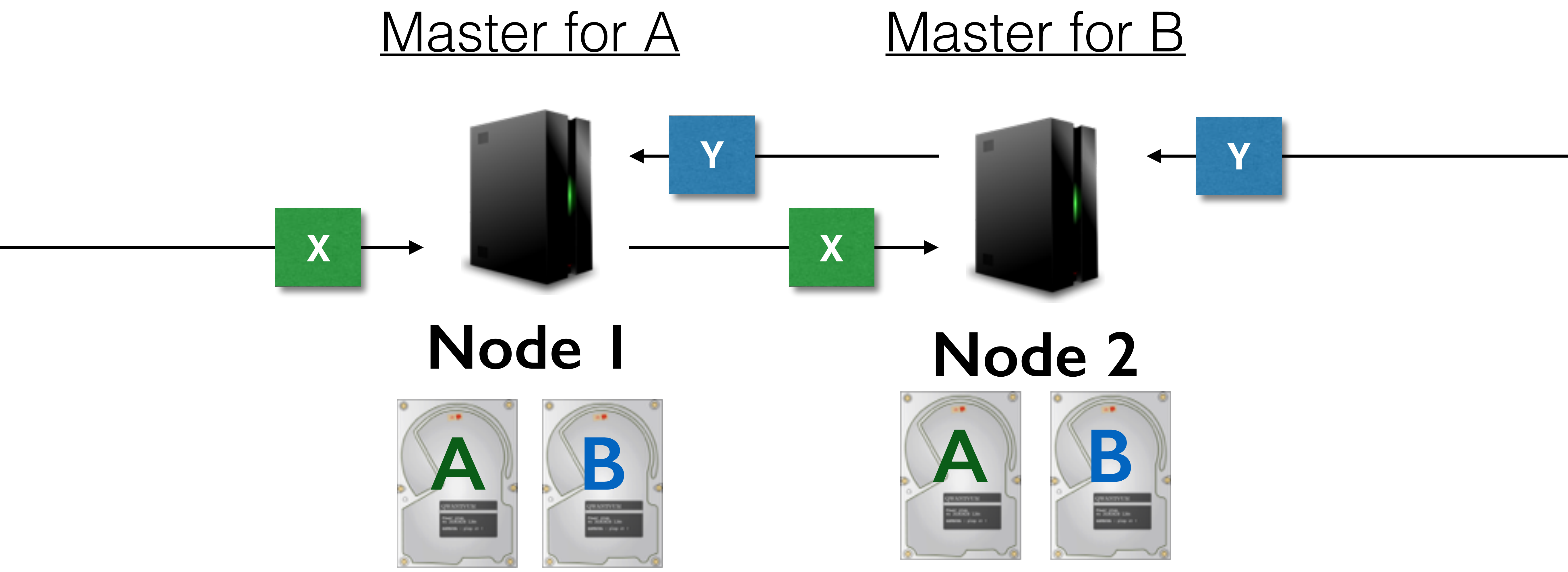
All data is consistent… but unavailable

C A P

CONSISTENCY or AVAILABILITY or PARTITION TOLERANCE

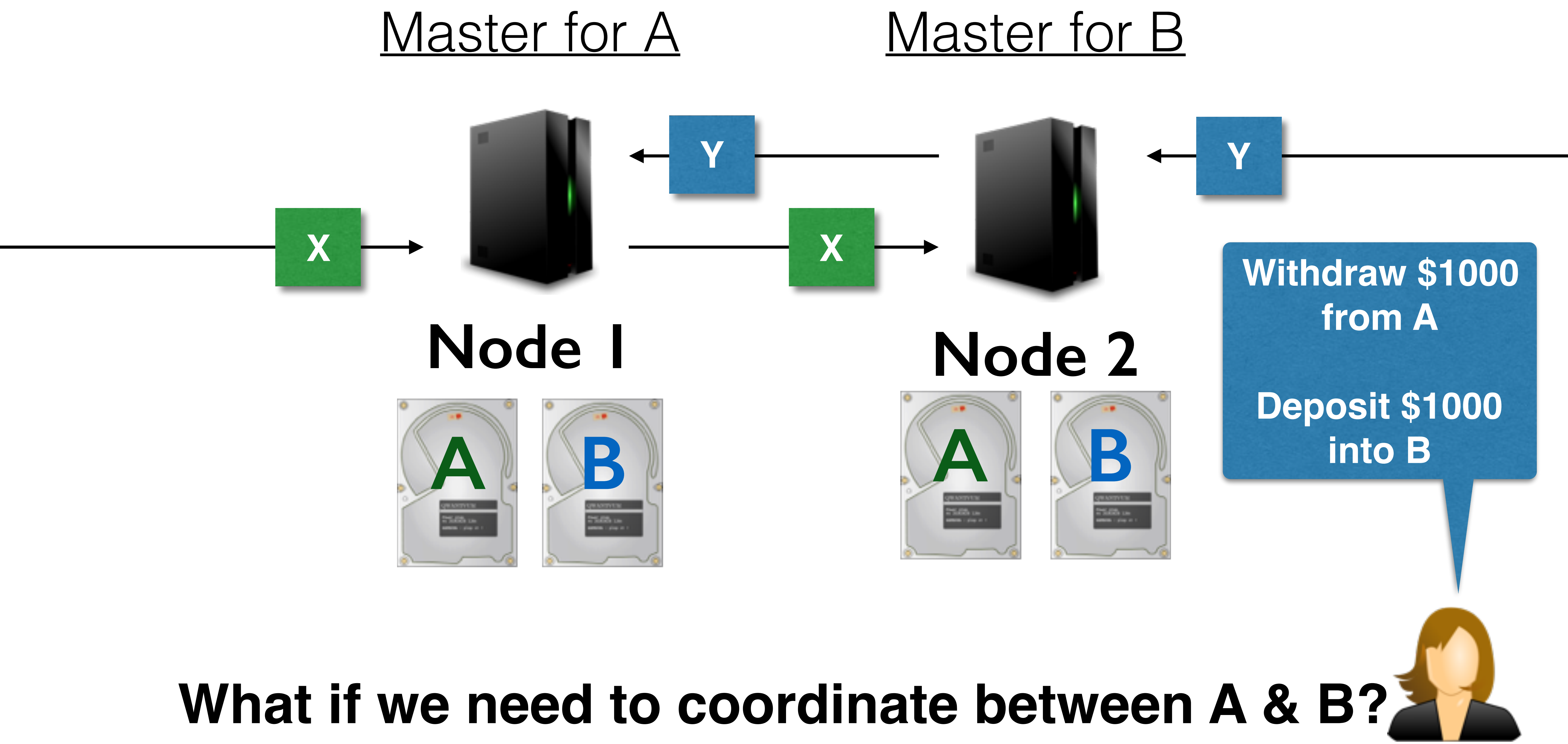
Traditionally: Pick any 2

Simpl-ish Consensus



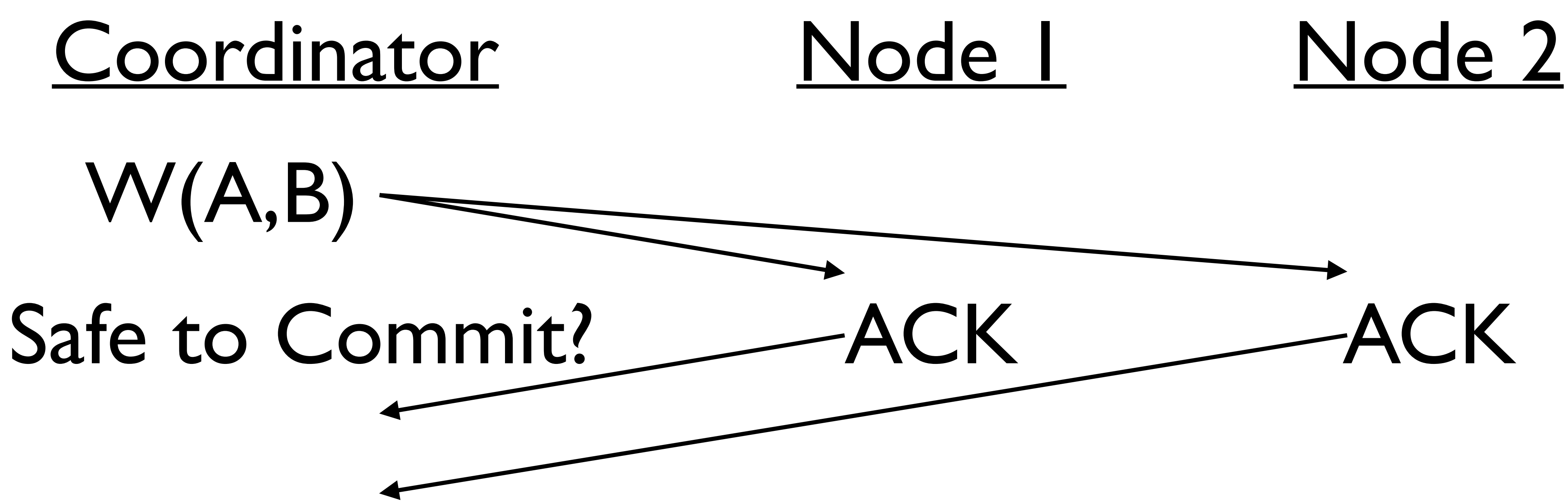
Node 2 agrees to Node 1's order for A.
Node 1 agrees to Node 2's order for B.

Simpl-ish Consensus



What if we need to coordinate between A & B?

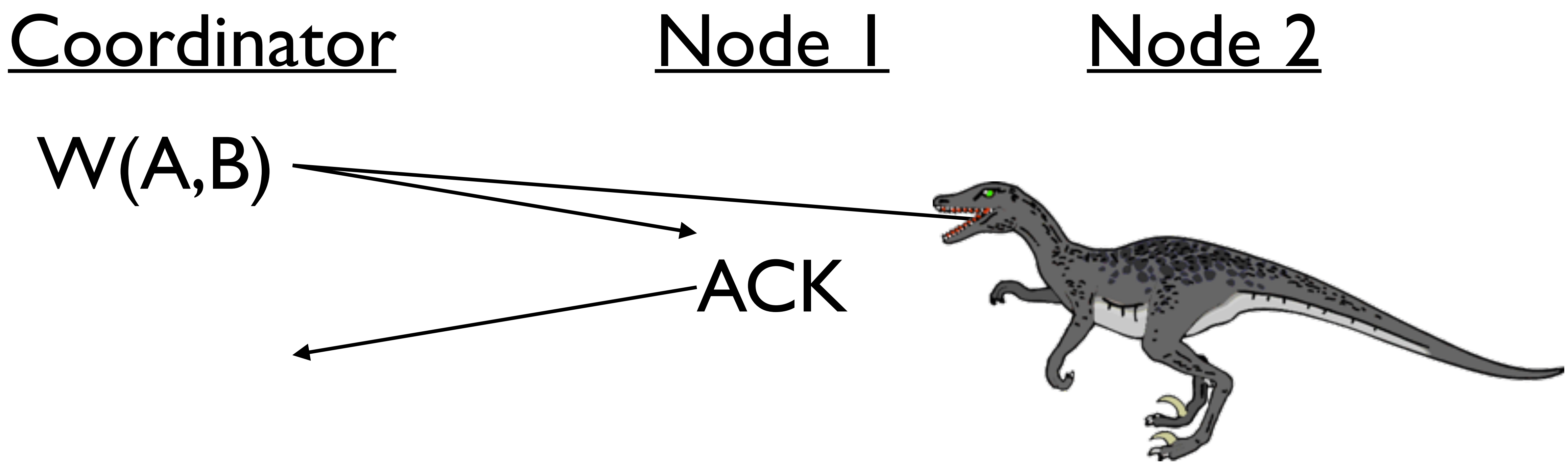
Naive Commit



Safe to Commit ?

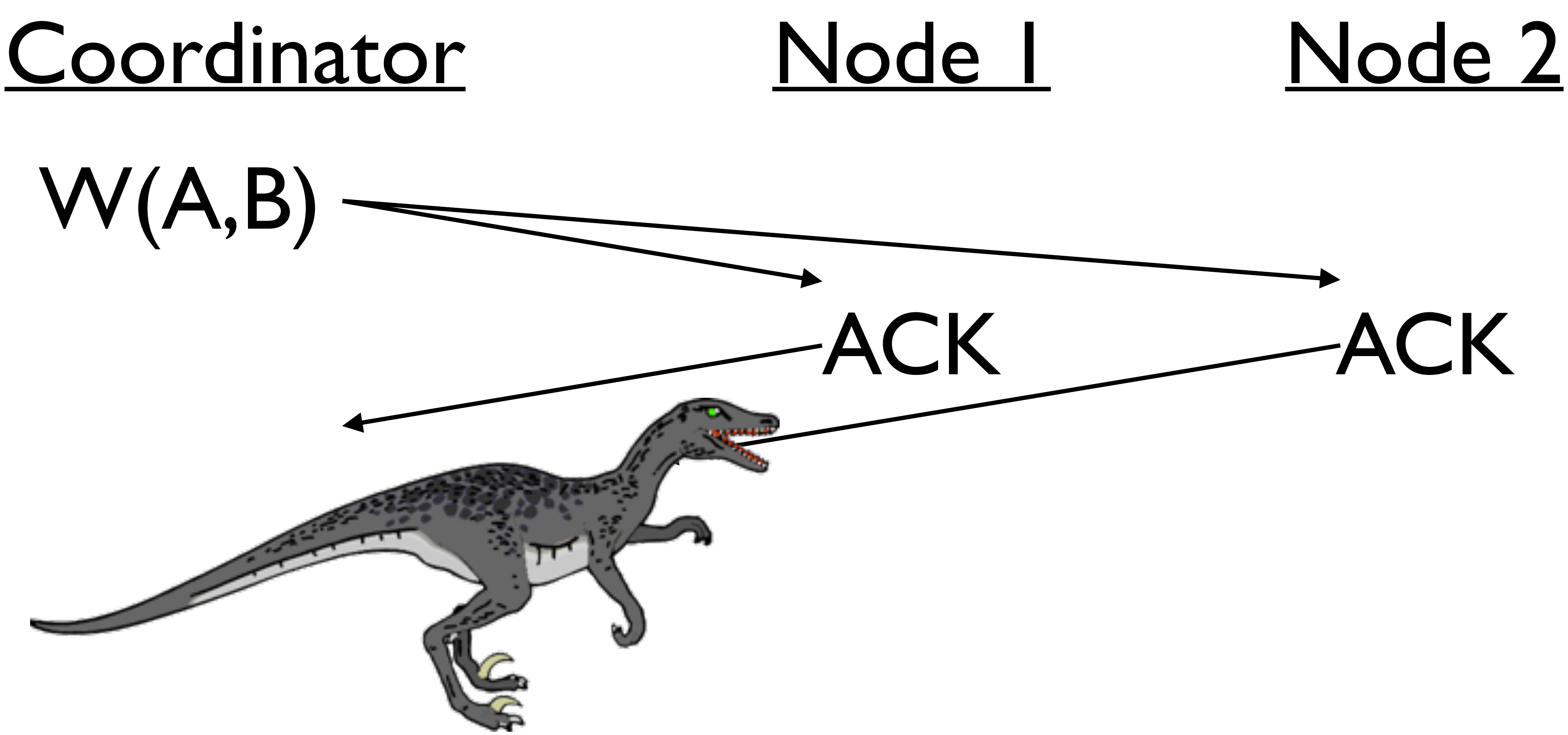
That packet sure does look tasty…

Naive Commit



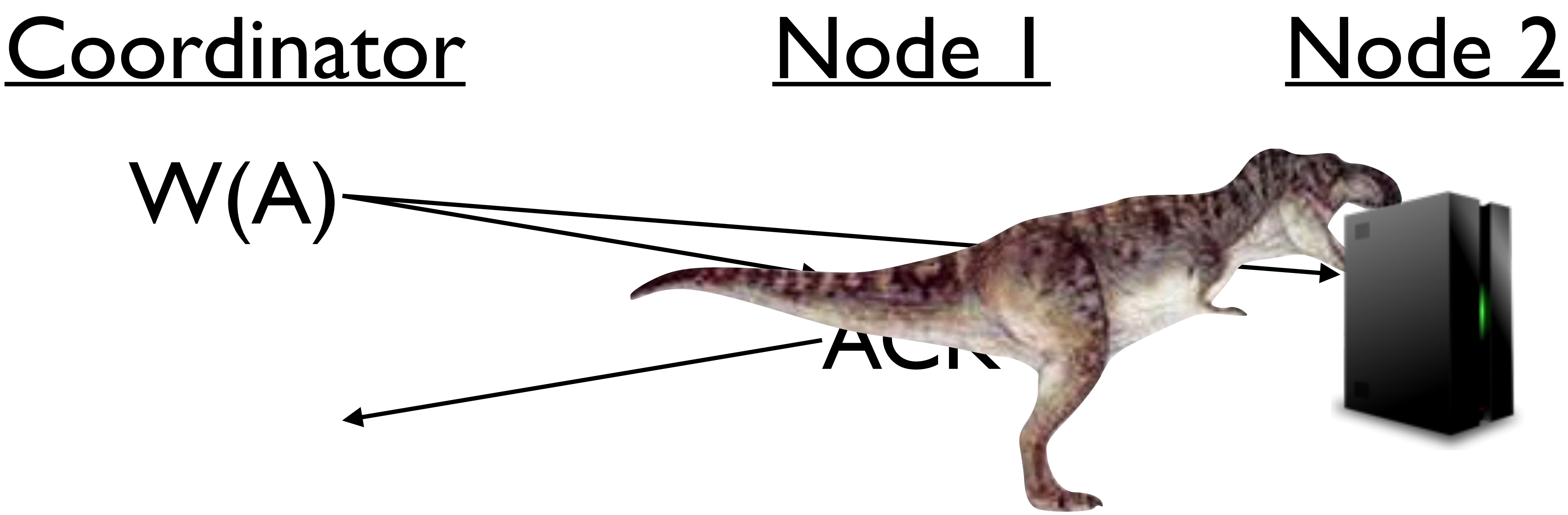
Is it safe to abort?

Naive Commit



What now?

Naive Commit



How do we know Node 2 even still exists?

2-Phase Commit

- One site selected as a coordinator.
  - Initiates the 2-phase commit process.
- Remaining sites are subordinates.

- Only one coordinator per xact.
  - Different xacts may have different coordinators.

2-Phase Commit

- Coordinator sends ‘prepare’ to each subordinate.
- When subordinate receives ‘prepare’, it makes a final decision: Commit or Abort.
    - The transaction is treated as if it committed for conflict detection.
    - The subordinate logs ‘prepare’, or ‘abort’
    - The subordinate responds ‘yes’, or ‘no’

2-Phase Commit

- If coordinator receives ‘no’ from <u>any</u> subordinate, it tells subordinates to ‘abort’.
  - Can treat timeouts as ‘no’s
- If coordinator receives ‘yes’ from <u>all</u> subordinates, it tells subordinates to ‘commit’
- In both cases, the coordinator first logs the decision and forces the log to local storage.

2-Phase Commit

- Subordinates perform abort or commit as appropriate (logging as in single-site ARIES)
- Subordinates ‘ack’nowledge the coordinator.
- The transaction is complete once the coordinator receives all ‘acks’.

2PC for Replication

- Optimization: We don't need 100% responses from replicas.
  - Replicas can be reconstructed from others.
  - Asserting 'preparedness' can be difficult.
- How much failure tolerance do we want?
  - We can tolerate N failures by waiting for N+1 responses during the 'prepare' phase.

Recovery

How do we recover from a (transient) <u>coordinator</u> crash in Phase 1?

What information/communication state is lost?

Can it be recovered?

(Does it need to be?)

Recovery

How do we recover from a (transient) coordinator crash in Phase 2?

What information/communication state is lost?

Can it be recovered?

Recovery

How do we recover from a (transient) subordinate crash in Phase 1?

What information/communication state is lost?

Can it be recovered?

Recovery

How do we recover from a (transient) subordinate crash in Phase 2?

What information/communication state is lost?

Can it be recovered?