CAP Theorem

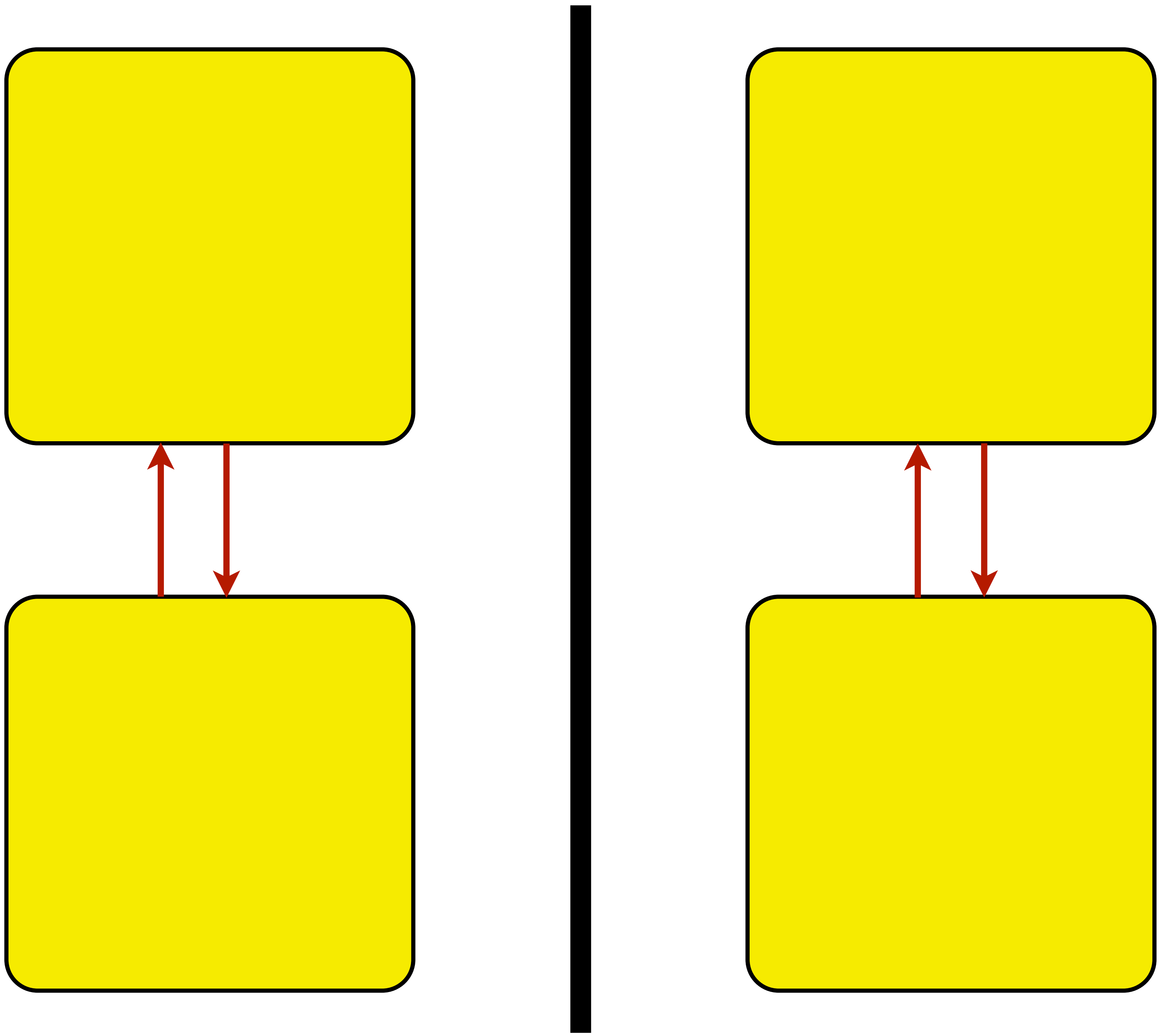
Definitions

Consistency: atomic, linearizable data items (each write appears to happen immediately across all nodes)

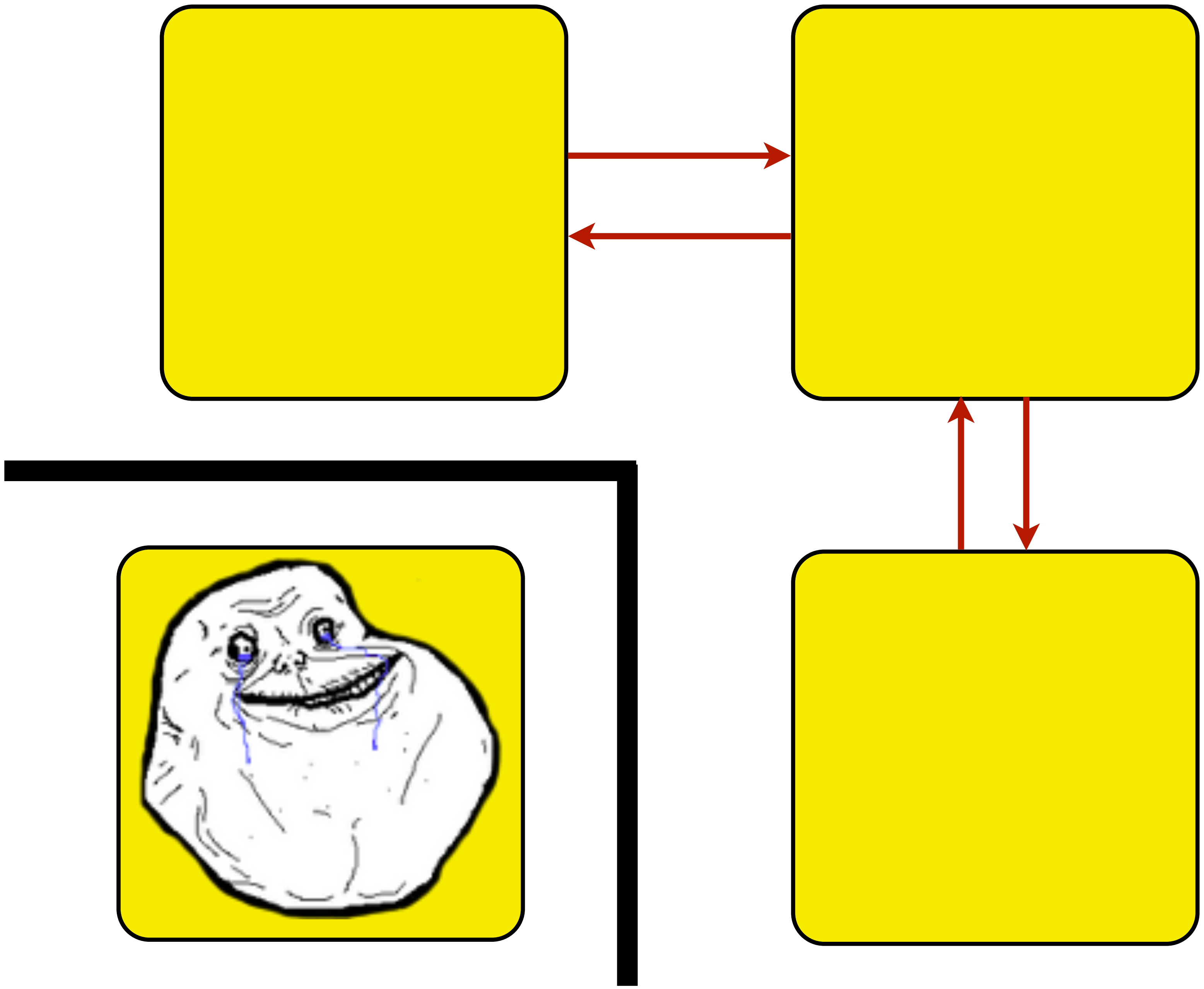
Availability: always get a response if your message goes through; no hanging

Partition tolerance: can lose messages (varying degrees)

Group Partition



Individual Partition

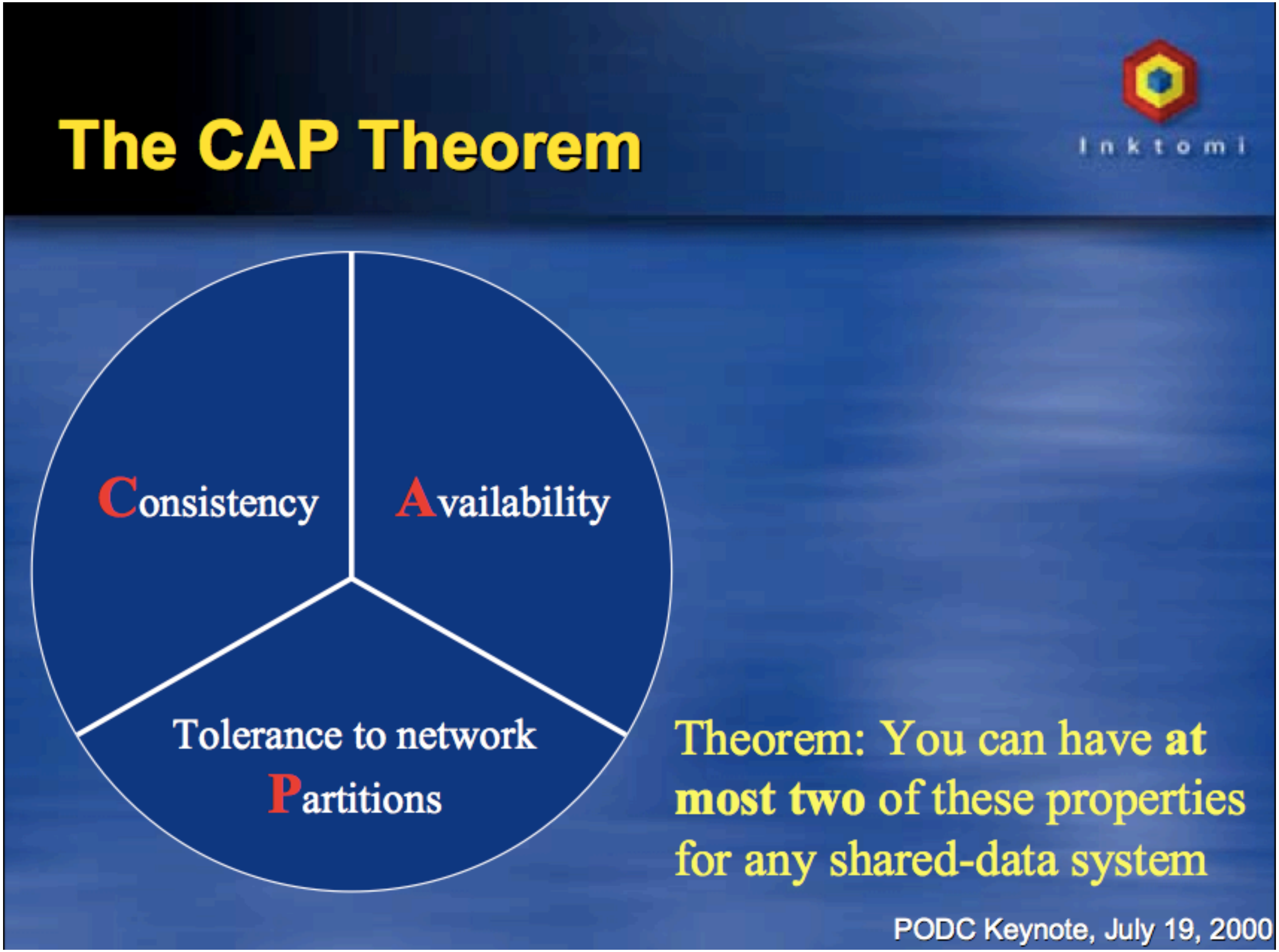


(Some) Related Work

- Coda Project (e.g., IEEE Trans. on Computers 1990): CMU, high availability in disconnected operation, sacrifice consistency
- The Bayou Project (e.g., SOSP 1995): Xerox PARC mobile device data synchronization, "anti-entropy" protocols
- "The dangers of replication and a solution" (Grey et. al, SIGMOD 1996): Lazy update propagation

Brewer's Work

- "Cluster Based Scalable Network Services" (SOSP 1997): Brewer and Inktomi, BASE principles
- "Harvest, Yield, and Scalable Tolerant Systems" (HotOS 1999): Brewer and Fox, actually describes **Strong CAP Principle**


The CAP Theorem
Inktomi
Consistency
Availability
Tolerance to network Partitions
Theorem: You can have **at most two** of these properties for any shared-data system
PODC Keynote, July 19, 2000

Do we believe CAP?

Gilbert and Lynch

- Provide formal proof of CAP
- Use asynchronous network model
  - No global clock
  - Agents act on local state and messages only

Theorem 1: It is impossible in the asynchronous network model to implement a read/write data object that guarantees the following properties:

- *Availability*
- *Atomic consistency*

in all fair executions (including those in which messages are lost)

Theorem 1, English

You can't have C, A, and P if you have arbitrary message delays and message loss.

Makes sense: how can two groups communicate updates if they can't communicate?

Key: availability requires that you return a value!

Corollary 1.1: It is impossible in the asynchronous network model to implement a read/write data object that guarantees the following properties:

- *Availability, in all fair executions,*
- *Atomic consistency, in fair executions in which no messages are lost*

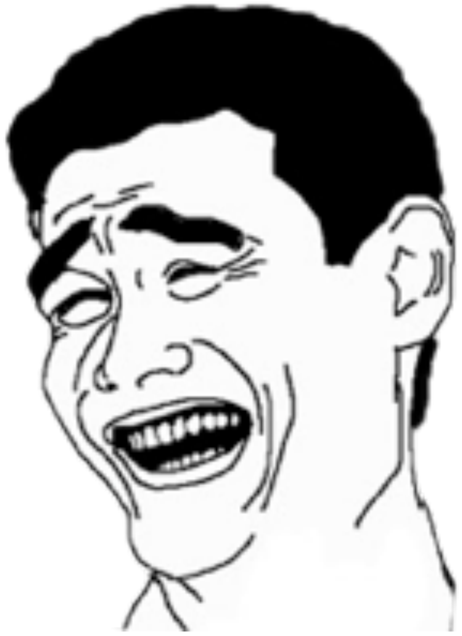
Corollary 1.1, English

Too bad! In the asynchronous model, we can't have C,A, and P even if we don't have partitions!

Makes sense: impossible to determine if a message has been delayed or if it's lost.

Chicken little: the sky (cloud?) is falling!!!

Can we do anything useful?!?



Of course;
Use proof by example

Recipe: C & P

```
def Handle_Request(socket):
   close(socket);
   return 0;
```

Recipe: C & P

```
def Handle_Request(socket):
   close(socket);
   return 0;
```

never accept writes!!!

never return anything!!!

(never available, so no wrong answers)

Recipe: C & A

Cake!

E.g., use a single master.

Recipe: A & P

```
def Handle_Read(socket):

    socket.write(init_value)

    close(socket);

    return 0;
```

```
def Handle_Write(socket):

socket.write(ACK);

//do nothing

close(socket);

return 0;
```

Recipe: A & P

```
def Handle_Read(socket):
    socket.write(init_value)
    close(socket);
    return 0;
```

```
def Handle_Write(socket):
socket.write(ACK);
//do nothing
close(socket);
return 0;
```

always return initial value

(never consistent, trivially available)

...what if we bound network delays?

...what if we bound network delays?

partial synchrony

Theorem 2: It is impossible in the partially synchronous network model to

implement a read/write data object that guarantees the following properties:

- *Availability*
- *Atomic consistency*

in all executions (even those in which messages are lost).

Theorem 2, English

Earthshaking: even with bounded message delays, if you lose messages arbitrarily, writes may not be propagated correctly and you'll get stale data

Key: availability requires that you return a value!

(Corollary 2.1): It is ***possible*** *in the partially synchronous network model to implement a read/write data object that guarantees the following properties:*

- *Availability, in all fair executions,*
- *"Variable, sometimes atomic consistency", in fair executions in which no messages are lost*

(Corollary 2.1), English

In absence of message loss, if you don't get an ack within 2*(max_msg_transit_time)+(time_spent_processing), then there was a partition!

Return consistent data in absence of partitions

Return inconsistent data with partitions, and detect this is happening

(Quickly,) Delayed-t Consistency

- Weaker consistency form
- In a nutshell, partially order non-concurrent operations
- Use knowledge of timeouts to determine if messages are lost, and use sequence numbers and centralized node to define ordering

Thoughts

- Do we need to have the formal proof in the paper?
  - Formalism is nice to have...
  - ...but it makes sense intuitively

Thoughts

- w.r.t. good design, systems people always say “it depends”
  - It’s nice to see a formalization of why “it depends”, and how “it depends” for once!

Thoughts

- Lots of work making CAP tradeoffs implicitly before "CAP Theorem" announcement
  - Was Brewer more perceptive than others?
  - Would we still have BASE systems like Dynamo and Cassandra without formal CAP theorem?
  - Who is the real Johnny Rotten here?

Thoughts

- What about "Weak CAP Principle"? (HotOS 1999)
  - "The stronger the guarantees made about any two of strong consistency, high availability, or resilience to partitions, the weaker the guarantees that can be made about the third."

Thoughts

- Daniel Abadi: PACELC
  - “if there is a partition (P) how does the system tradeoff between availability and consistency (A and C); else (E) when the system is running as normal in the absence of partitions, how does the system tradeoff between latency (L) and consistency (C)?”
  - http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

End of Slides