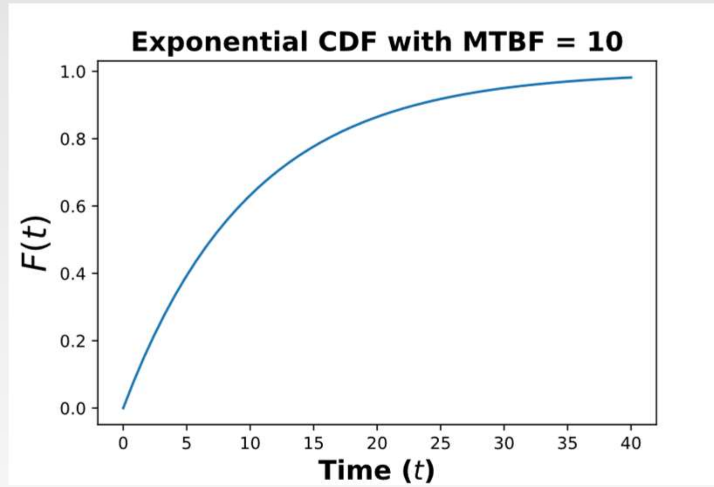
Statistical models

Analysis of failure and recovery algorithms assume that failures occur based on a probabilistic process

- Closed-form description
- Failures are independent
  - Not true(bathtub model, cascading failures)

As time increases, we expect the probability of a failure to increase as well

- **Cumulative Distribution Function** (CDF): Probability of failure before time t



Exponential CDF:

$F(t) = 1 - e^{-\lambda t}$, where λ is the failure rate λ = 1 / MTBF

Weibull CDF:

$F(t) = 1 - e^{(-\lambda t)^k}$, can be used to model

- decreasing failure rate ($k < 1$)
- constant failure rate ($k = 1$)
- increasing failure rate ($k > 1$)

Probability of failing between time and t_1 and t_2 is $\mathrm{F}(t_2) - \mathrm{F}(t_1)$

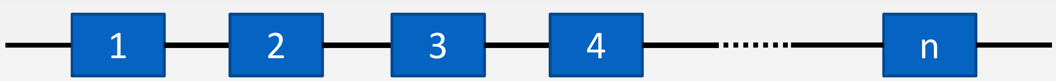
Modeling system reliability

The reliability function $R(t)$ models a single component or subsystem

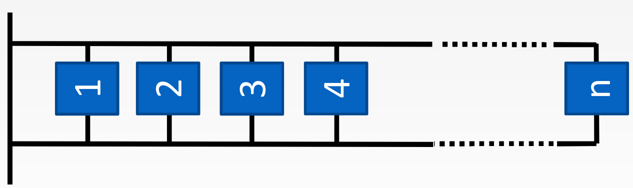
$$R(t) = P(X > t) = 1 - F(t)$$

What if multiple components are connected?

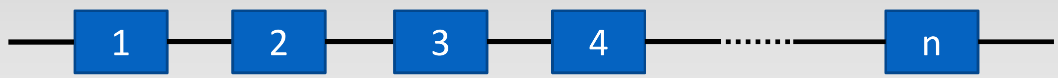
Series:



Parallel:



Series system



$$R_{series}(t) = \prod_{i=1}^{n} R_i(t)$$

$R_i(t)$ is the reliability of component i

All components need to survive for the system to function

For exponentially distributed failures:

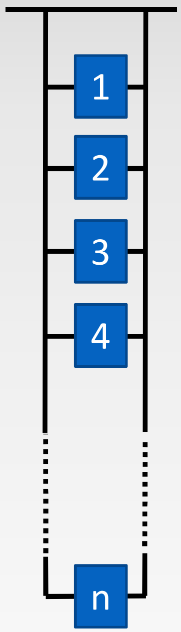
$$R_{series}(t) = \prod_{i=1}^{n} R_i(t) = e^{-\sum_{i=1}^{n} \lambda_i t} = e^{-\lambda_{system} t}$$

Parallel system

System with spares

Assume that when a failure occurs, the spare assumes responsibility immediately

$$R_{parallel}(t) = 1.0 - \prod_{i=1}^{n}(1.0 - R_i(t))$$

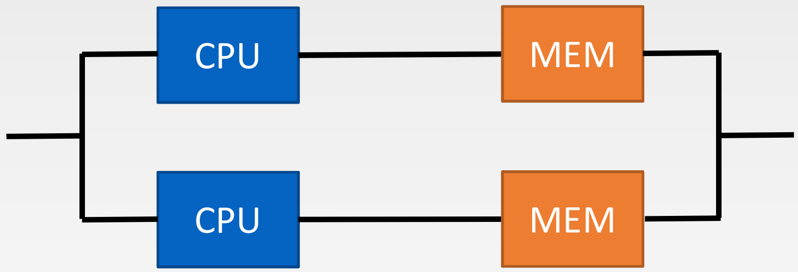


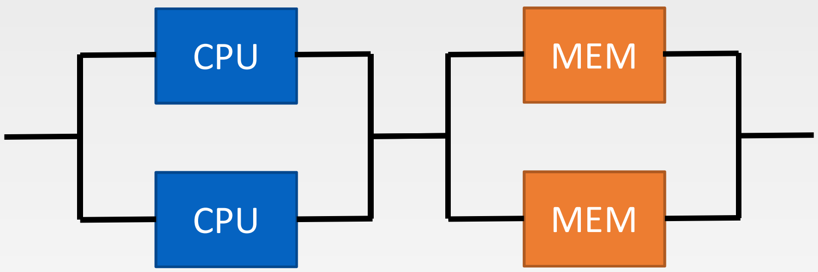
For the system to work only one component needs to operate correctly

- Probability of module i to survive: R_i
- Probability module i does not survive: $(1 - R_i)$
- Probability of no modules survive: $(1 - R_1)(1 - R_2) \dots (1 - R_n)$

Example

What is the reliability of each system?





Homework 1

Outline

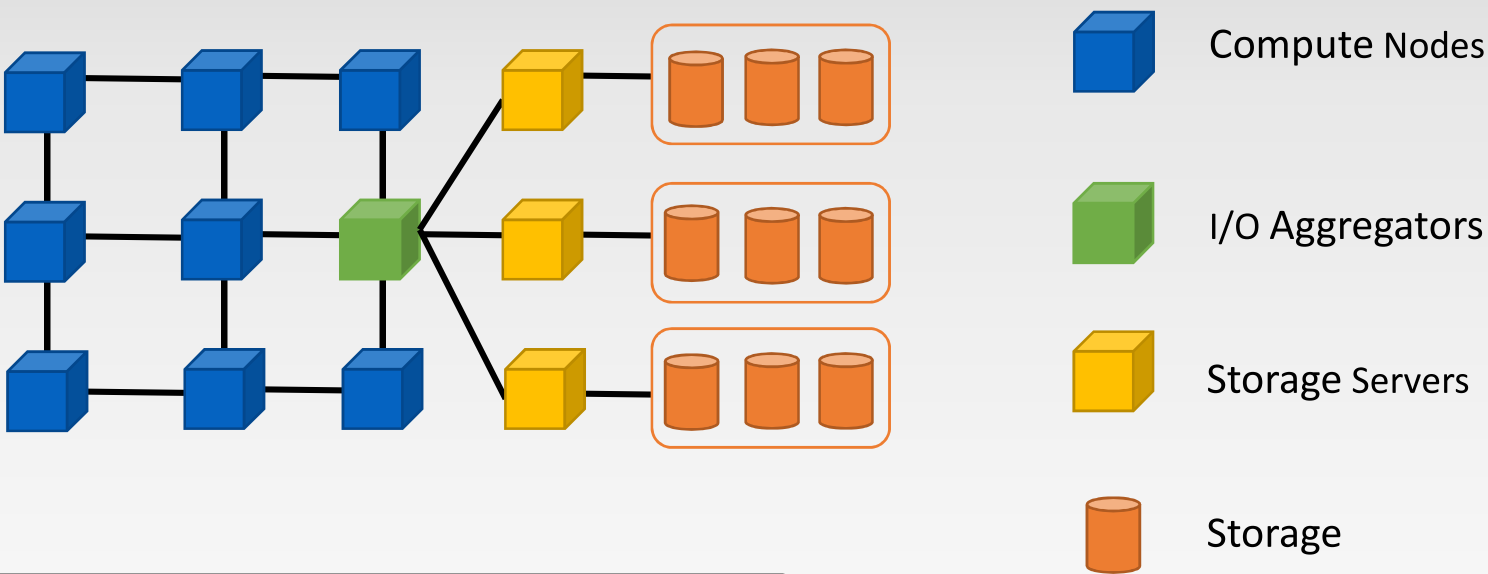
Taxonomy of common terms

Modeling reliability

Basics of failure detection in a distributed environment

"Seeing is believing"

Our "machine"



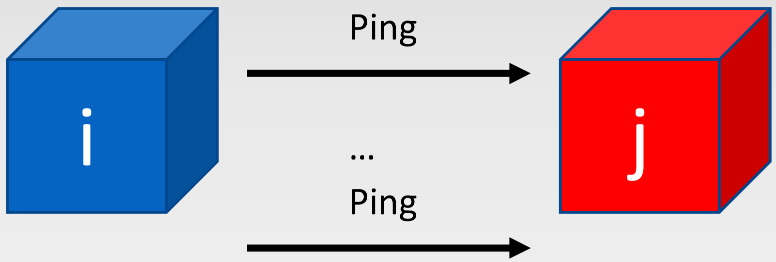
How do we know when something goes wrong?

Ping-ack

Let's consider two nodes:

Ping-ack

One node crashes

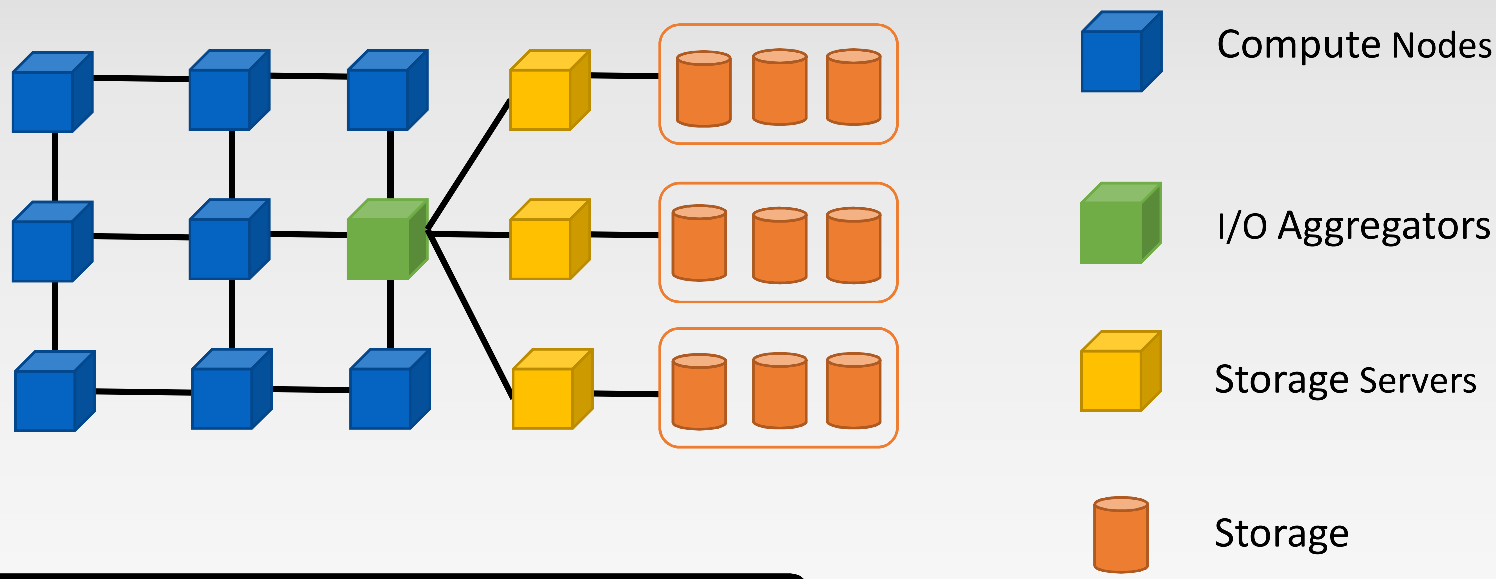


...

Node i queries node j every T time units

Time out window should be some multiple of the message round trip time

Our “machine”



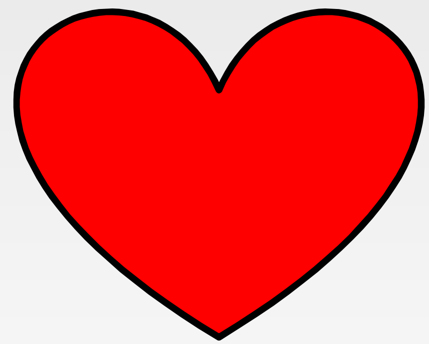
How do we know when something goes wrong?

This “machine”

How do we know when something goes wrong?

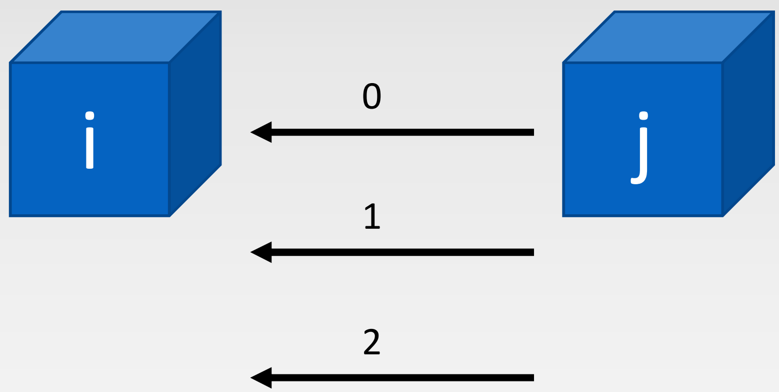
How do we know a human is ALIVE?

HEARTBEATS



Heartbeat

Let's consider two nodes:



Node j sends a heartbeat message containing a sequence number every T time units

If i has not received a heartbeat within $3 * T$ time units, assume node j has failed

Evaluation of the failure detectors

Completeness: every failure is eventually detected (no misses)

Accuracy: every detected failure corresponds to a crash (no mistakes)

Completeness and Accuracy can be guaranteed 100% in a synchronous

Why can Completeness and Accuracy never be guaranteed simultaneously on asynchronous systems?

Completeness and accuracy in asynchronous systems

Impossible to have both due to arbitrary message delays and message losses
- Dropped heartbeats/acks cause the appearance of failure

Message delays and losses are impossible to distinguish from a faulty process

Would you rather have 100% completeness or 100% accuracy?

Why does Ping-ack and heart beating satisfies completeness but not accuracy

Heart beating

Although it can not satisfy accuracy and completeness at the same time on asynchronous systems, it is still widely used to detect failure of

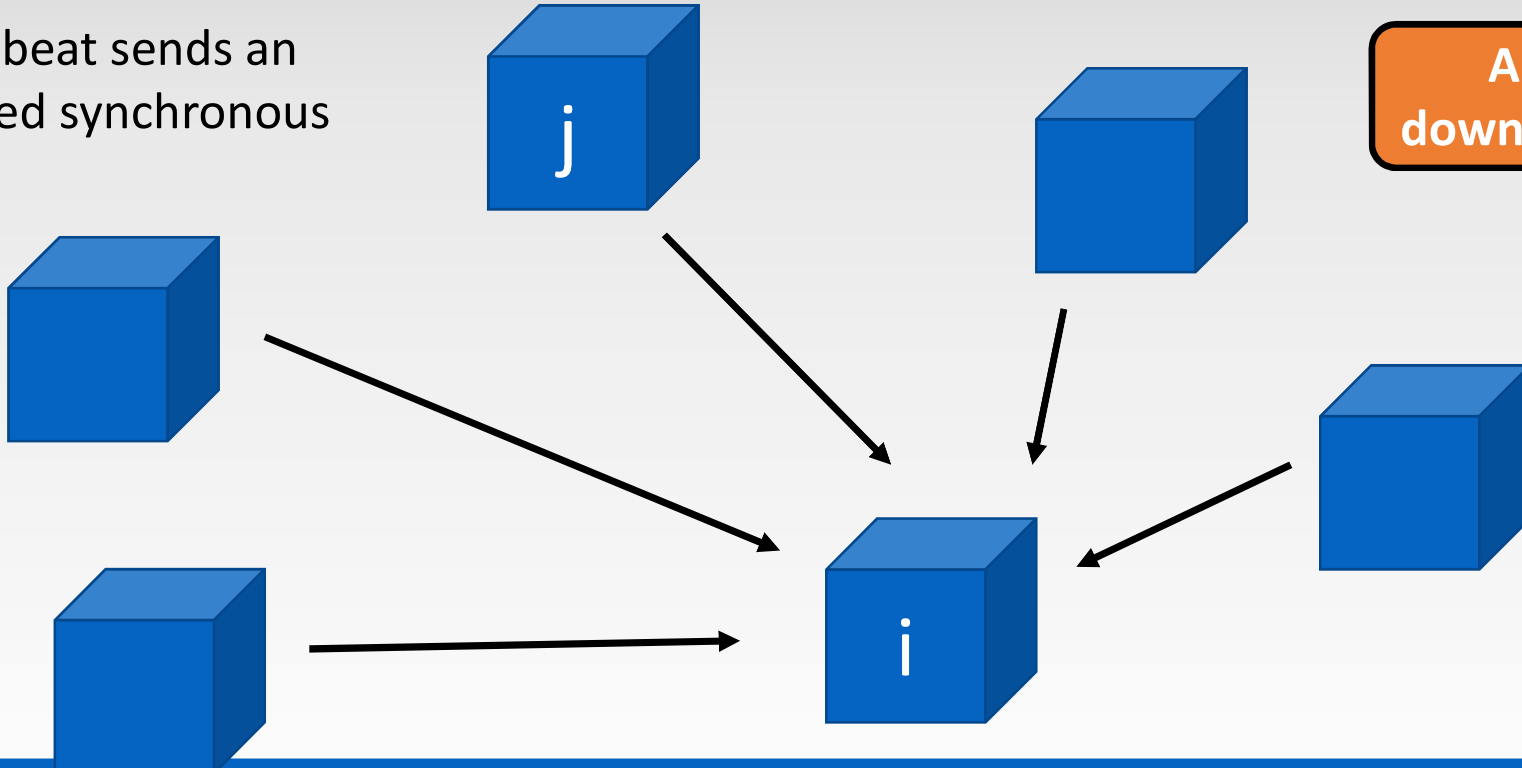
- Processes
- Nodes
- Blades
- Cabinets

How would you implement heart beating?

Let’s now expand out system beyond 2 nodes and examine how heart beating can be implemented

Centralized heart beating

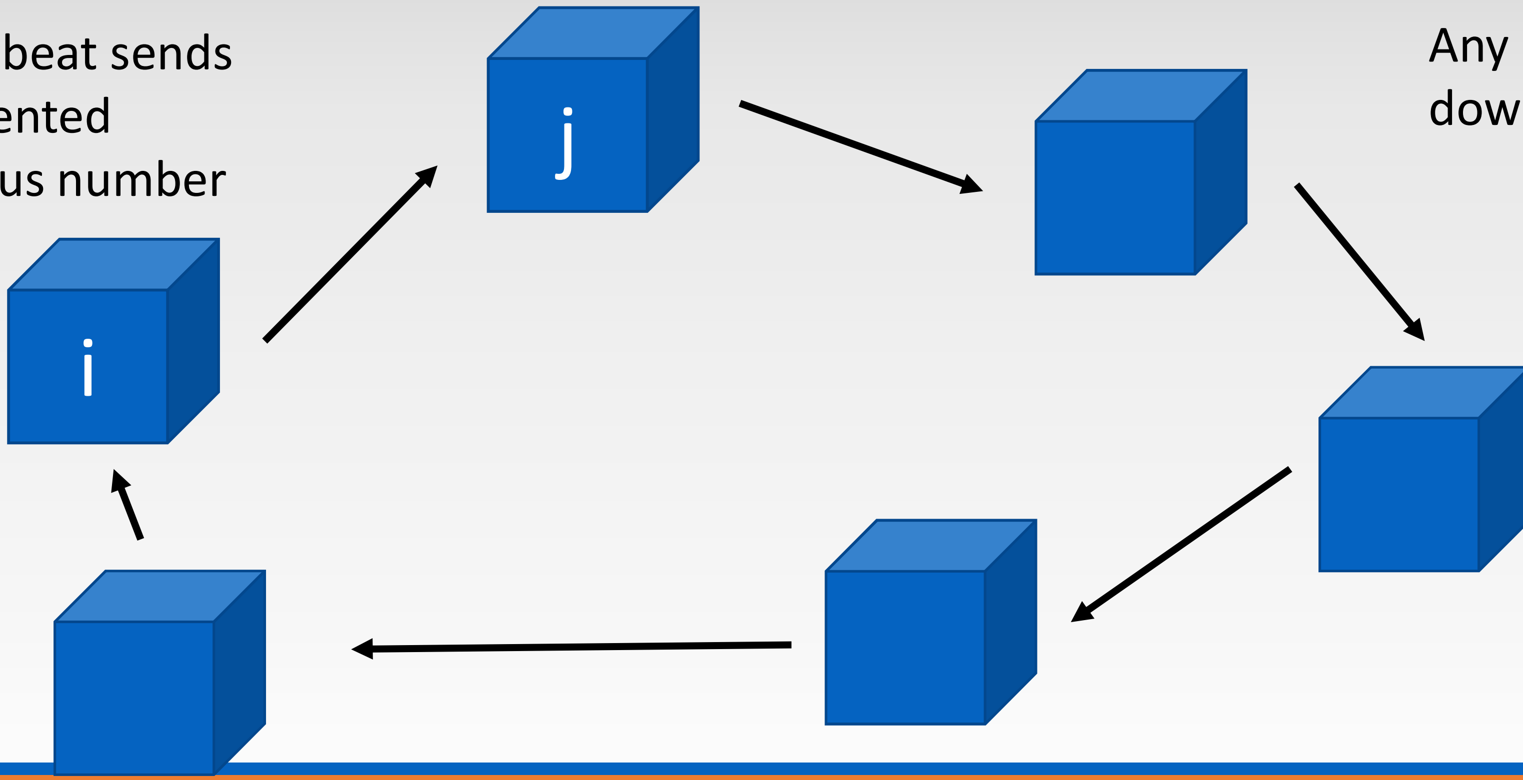
Each heartbeat sends an incremented synchronous number



Any downsides?

Ring heart beating

Each heartbeat sends an incremented synchronous number

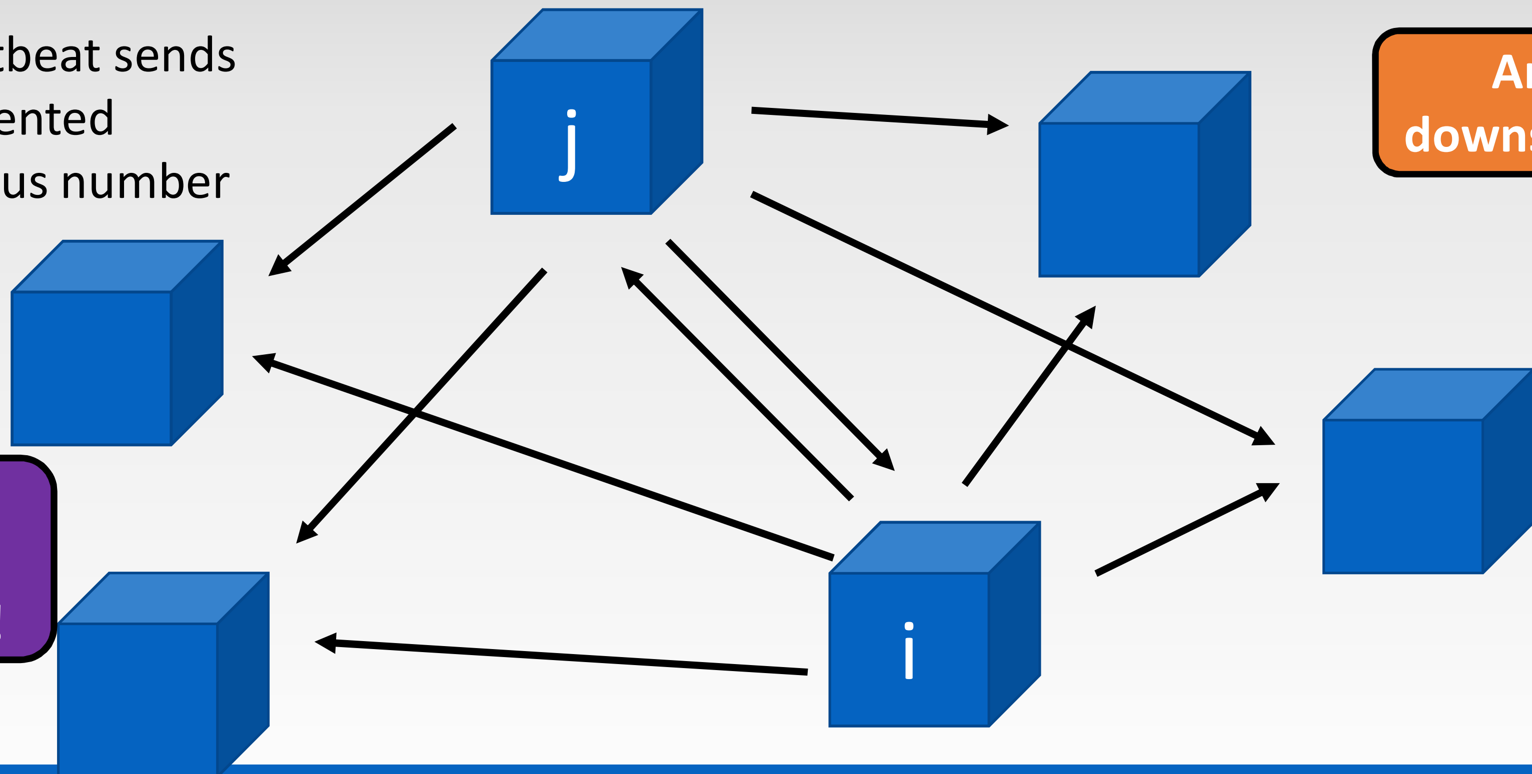


Any downsides?

No single point of failure!

All-to-All heart beating

Each heartbeat sends an incremented synchronous number



Any downsides?

Everyone knows everything!

Evaluation metrics

Bandwidth: Fewer messages the better

- Limits impact on application performance

Detection Time: the shorter the latency detection after a crash the better

Scalability: How does bandwidth and detection time change as the system grows

Accuracy: Few false positives and no false negatives

Summary

Defined terms that will be used throughout the semester
- It means nothing if you can not communicate

Discussed metrics used to evaluate system reliability
- Look for a homework on this (out before next class)

Explored common techniques to failure detection on distributed machines

Next time we will discuss reliability issues of current petascale systems and explore predictions for exascale

References

ECE 542 University of Illinois Lecture Slides
- https://courses.engr.illinois.edu/ece542/sp2015/

CS 425 University of Illinois Lecture Slides
- https://courses.engr.illinois.edu/cs425/fa2016/

“Addressing failures in exascale computing” – Marc Snir et al.
- https://www.mcs.anl.gov/papers/P5022-0913.pdf

“Basic concepts and taxonomy of dependable and secure computing” -- Avizienis et al.
- https://ieeexplore.ieee.org/document/1335465/

Schadenfreude
- http://www.datacenterknowledge.com/archives/2010/05/13/car-crash-triggers-amazon-power-outage/