51.505 – Foundations of Cybersecurity

Week 4 - Distributed Systems

Created by **Martin Ochoa** (2017)
Modified by **Jianying Zhou** (2018)

Last updated: 11 Sept 2018

Recap

- Questions on last week's exercises?

Recap: CIA

Traditionally security is defined in terms of Confidentiality, Integrity and Availability of system data and resources.

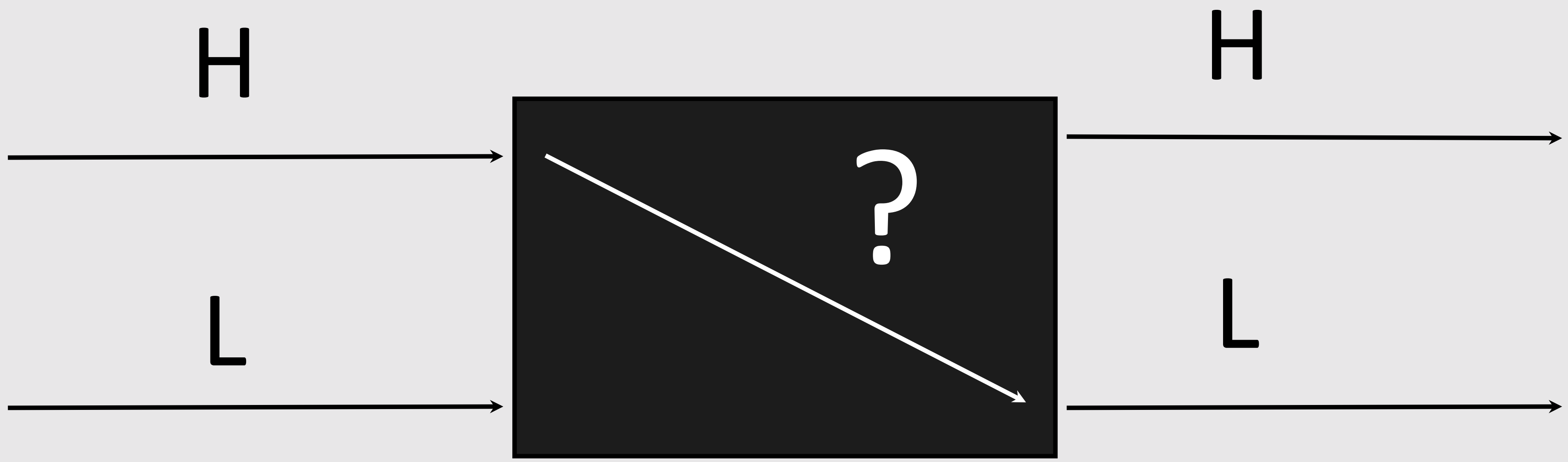
- What does exactly mean for something to be confidential/integer/available?

- How can we prove/disprove that a system is secure?

- Is there a rigorous definition?

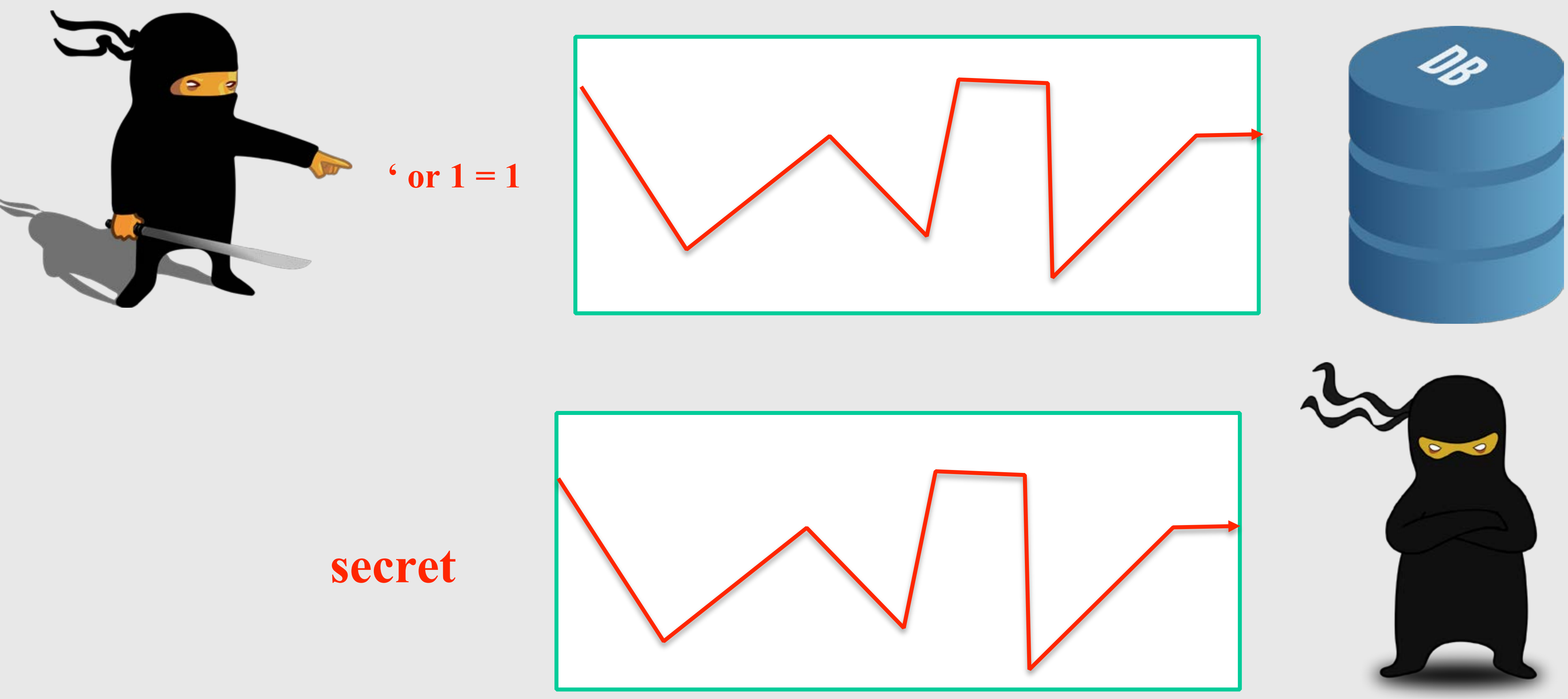
Discussion

- Why is non-interference interesting?
  - ✓ It represents a well defined notion of confidentiality and integrity.
  - ✓ It covers both explicit flows and implicit flows.



Unwanted Flows

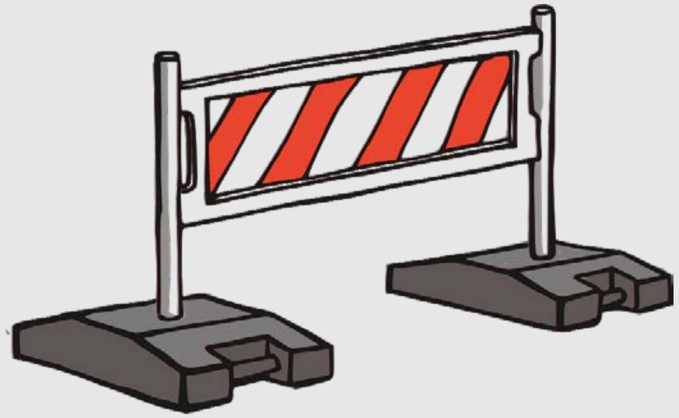
- Exploiting a vulnerability that alters data is an <u>integrity</u> violation.



- An attack that leaks information violates <u>confidentiality</u>.

Access Control and beyond

- How to enforce information flow in practice?
- Historically (in practice) the obvious thing to do is to restrict the access of certain parties to certain resources.



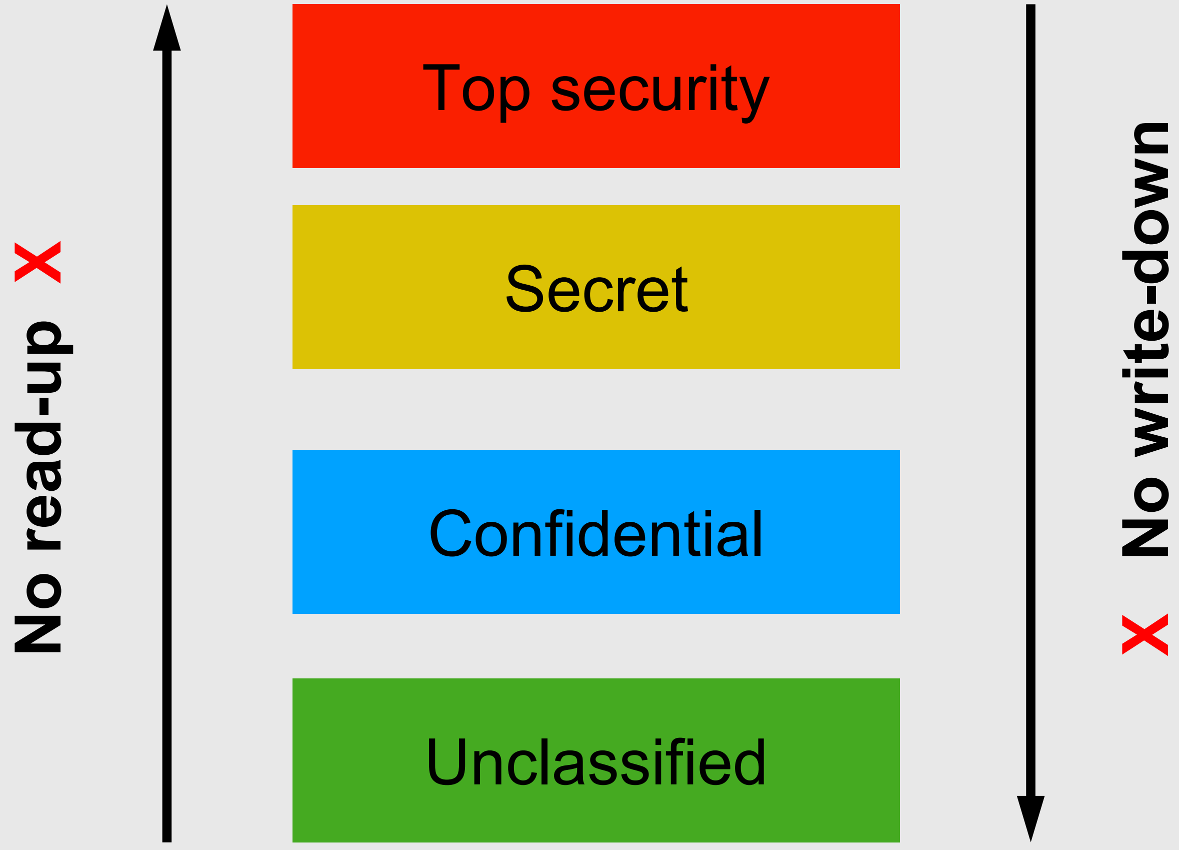
Access Control Matrix

	File 1	File 2	Process 1	Process 2
User A	read, write, own	read	read, write, execute, own	write
User B	append	read, own	read	read, write, execute, own

- This can be basically represented by a matrix where a set of subjects are granted privileges on a set of objects or resources.
- Enough to provide information flow guarantees?

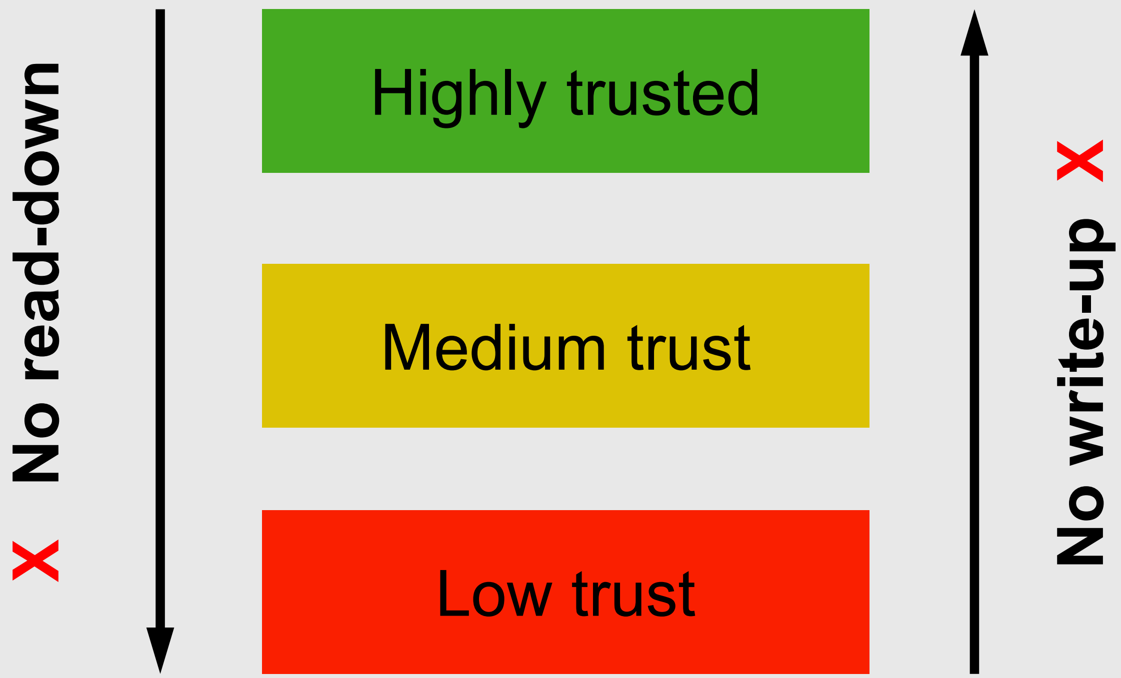
Bell-LaPadula (Confidentiality)

- Essentially: No read-up, no write-down.



Biba (Integrity)

- Essentially: No read-down, no write-up.



Are we done?

- So far we have covered some fundamentals in terms of confidentiality and integrity:

  ✓ *Information flow*: models confidentiality and integrity.

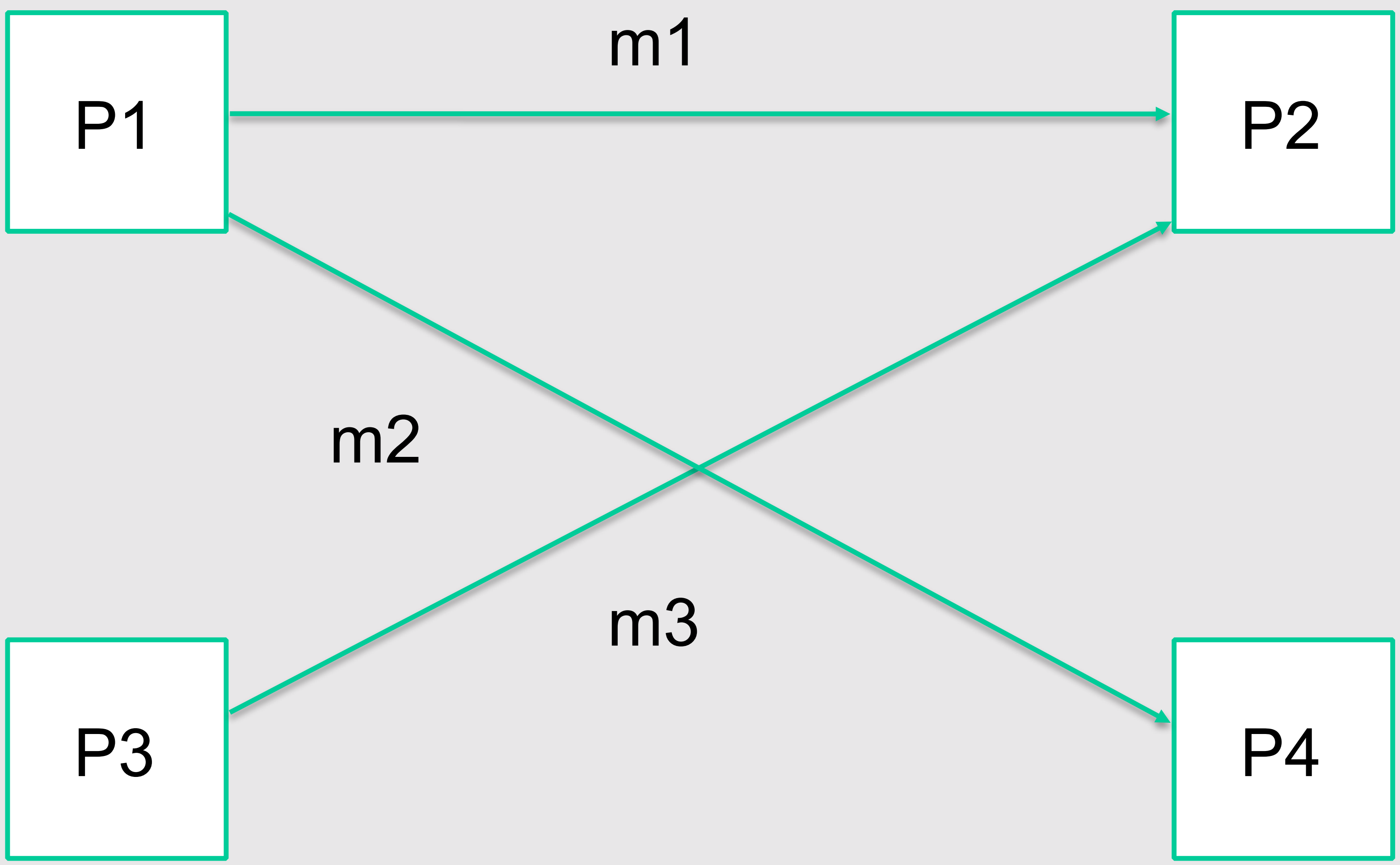
  ✓ *Access control*: rules actions that principals can take, in order to prevent confidentiality and integrity issues.

- What about **availability**?

Motivation

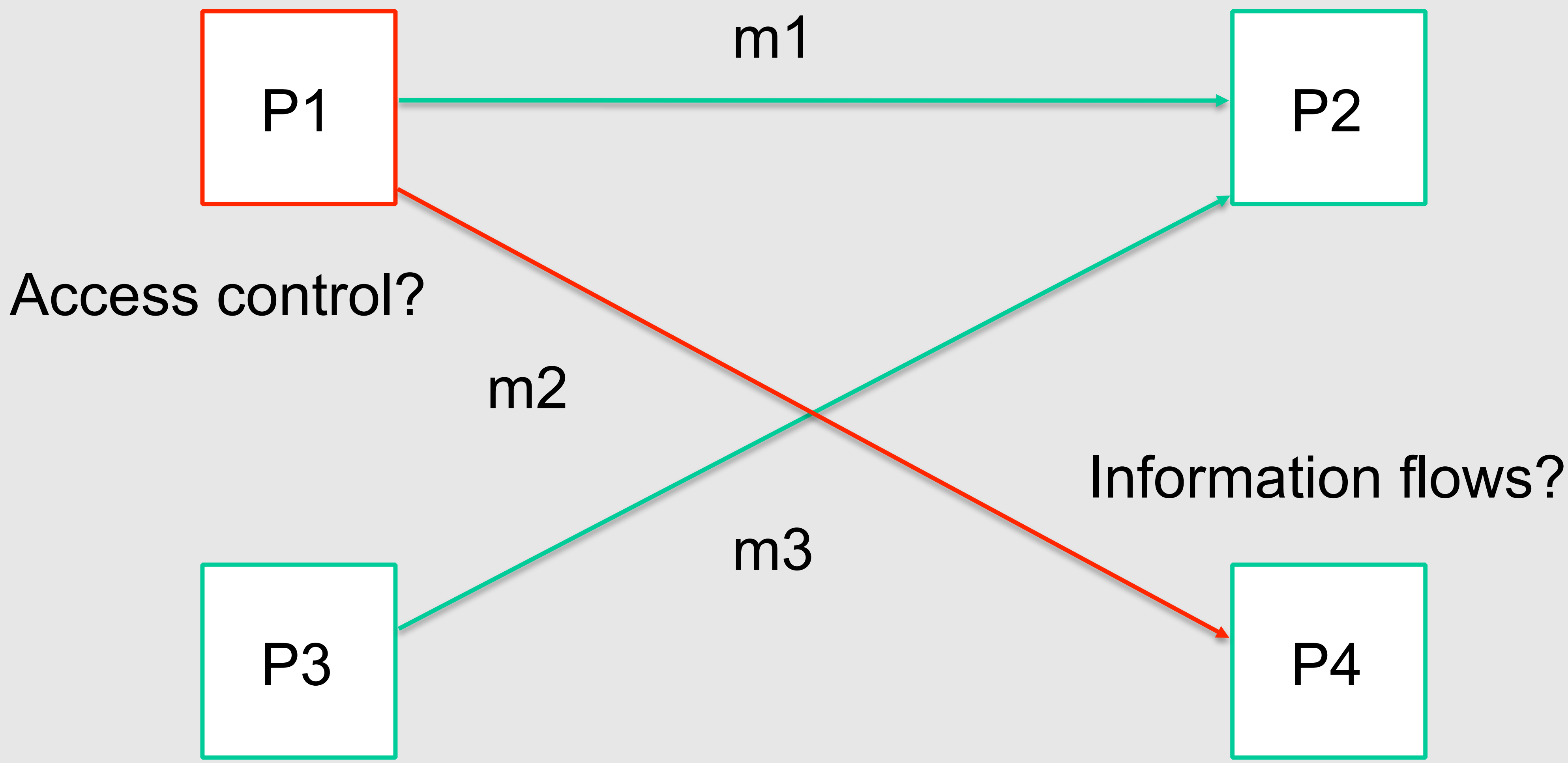
- What are distributed systems?
  - ✓ Collection of autonomous processes that communicate with each other to achieve a common computing task.
  - ✓ Examples:
    - ❖ Any computer network (Internet, LAN)
    - ❖ Multi-player online games
    - ❖ Industrial control systems
    - ❖ Multi-threading and virtualisation
- Why are they relevant?
  - ✓ Increasingly pervasive (multi-core computing)
  - ✓ In the future even more: Internet of Things, blockchains

Distributed Systems



- Each process has a <u>local</u> state.
- Processes behave <u>non-deterministically</u>. (Why?)

Distributed Systems



- Each process has a local state.
- Processes behave non-deterministically. (Why?)

Security Issues

- Everything we have discussed so far (Information Flows, Access Control)
- Confidentiality and integrity over channels (we will talk about this later)
- Focus today:
  - ✓ Concurrency
    - ❖ Race conditions
    - ❖ Deadlocks
    - ❖ Secure time
  - ✓ Fault tolerance and recovery
    - ❖ Denial of Service

Concurrency

- As you might have seen in other lectures, concurrency is the source of many problems in software systems.

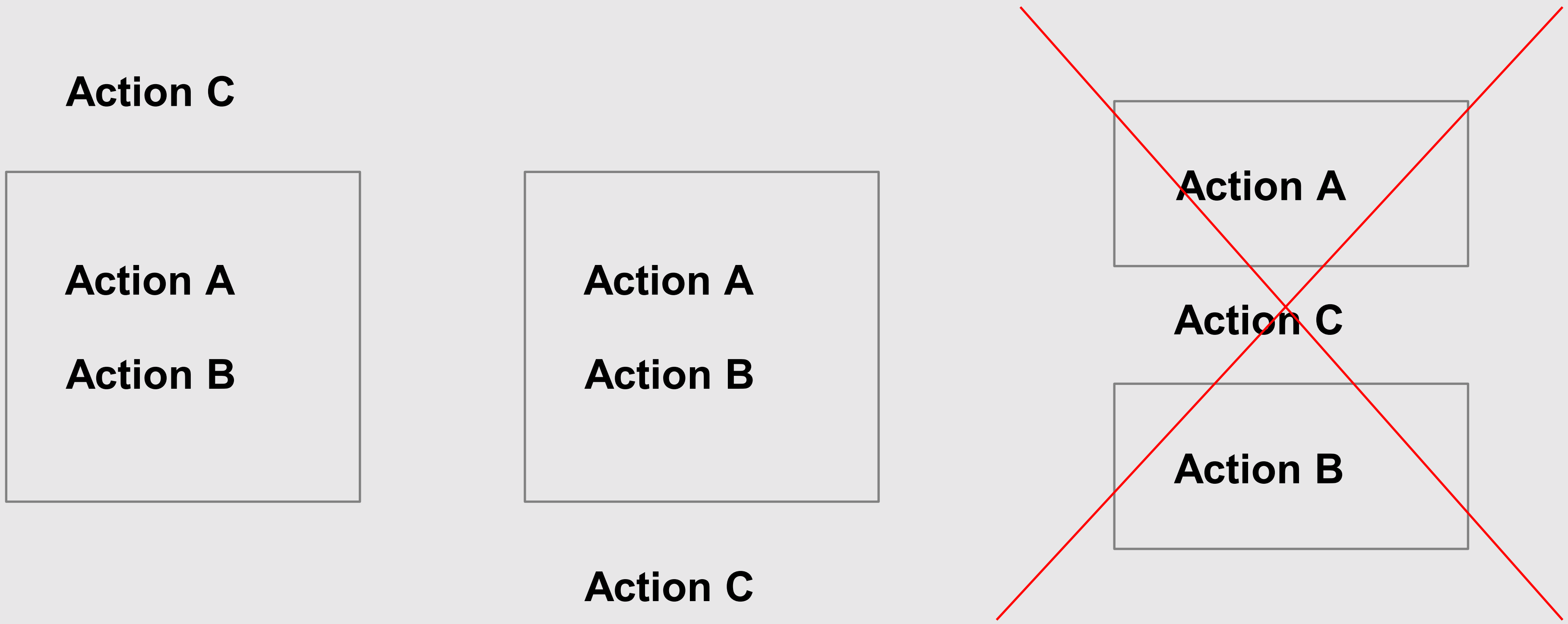
  ✓ It is challenging to foresee all possible interleavings between concurrent processes, and therefore undesirable states might be reached.

  ✓ *In operating systems*: interleavings determined by scheduler. Scheduling might be non-deterministic.

  ✓ *In networks*: interleavings determined by nodes, which typically behave in a non-deterministic fashion.

  ✓ *In critical systems*: sensors send information about environment. Some events will be non-deterministic (pressure of a valve greater than a certain value, speed of wind etc).

Concurrency



Common concurrency issue: *Atomicity* of a block of operations is violated.

Concurrency & Security

In security generally referred to as TOCTOU (time of check, time of use) vulnerabilities (or race conditions):

Revoke A

```
p = Check_permission(A)

   If p{
   }
```

```
p = Check_permission(A)

   If p{
   }
```

Revoke A

```
p = Check_permission(A)
```

Revoke A

```
   If p{
   }
```

?

For instance, certificates (will come back to this).

Concurrency & Security

In security generally referred to as TOCTOU (time of check, time of use) vulnerabilities (or race conditions):

File = X

```
p = Check_permission(File)

    If p{
    Read(File) }
```

File = X

```
p = Check_permission(File)
```

File = Y

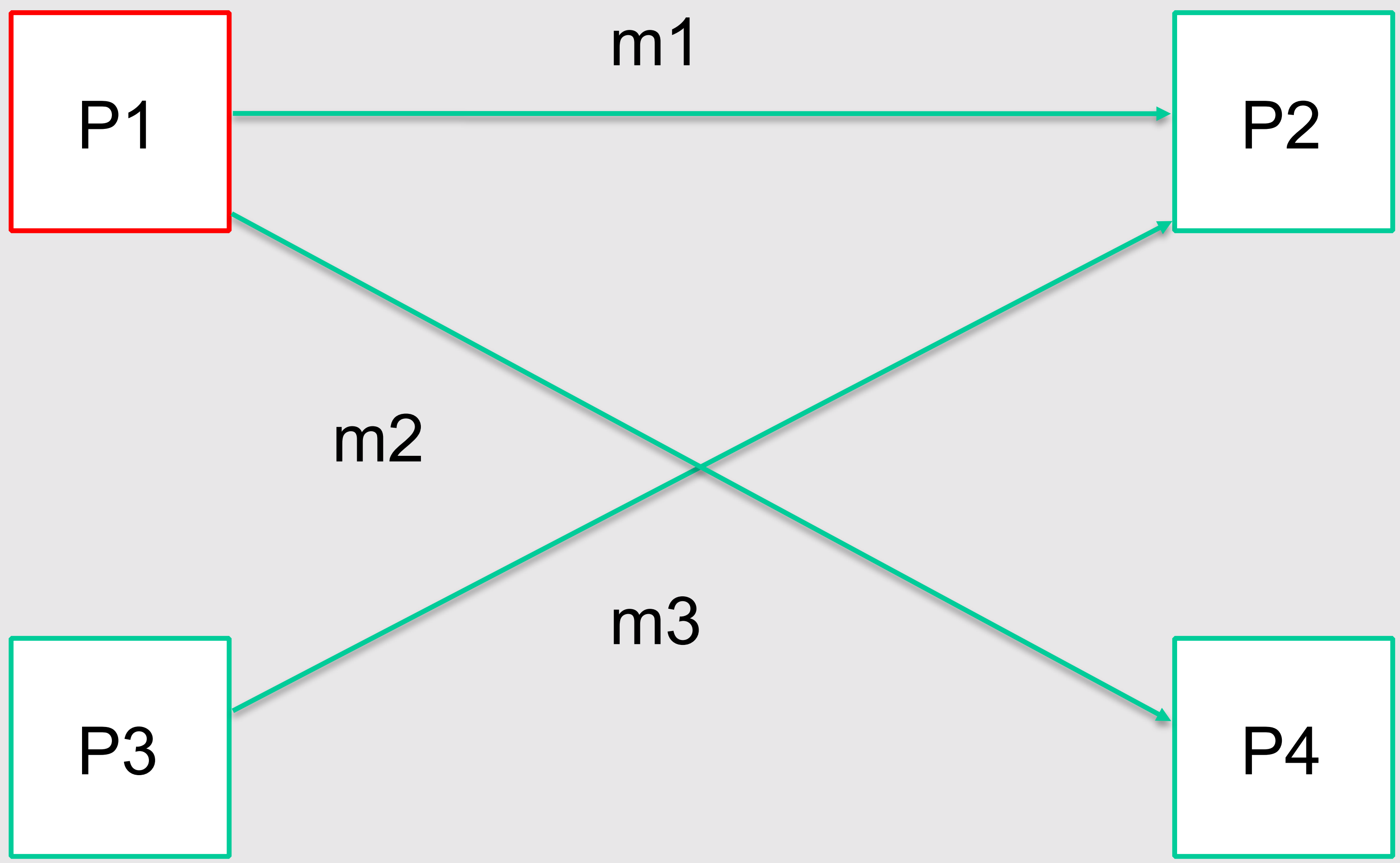
```
If p{

Read(File)}
```

?

For instance, use symbolic links to change the file pointer.

Consistent Global State



- Changes in local state of processes might affect the security of the complete system and thus must be propagated.
- This is challenging if number of nodes is high and real-time constraints are important.

State Propagation

- Credit card numbers are stolen every day.
- Banks maintain lists of stolen credit card numbers (globally this list has millions of records).
- For costs/functionality reasons, this state (all stolen credit cards) is not propagated worldwide to all merchants, neither they check each card used with the issuing bank.
- Therefore:
  - ✓ If amount is small, no immediate check.
  - ✓ If card is local, check.
  - ✓ If amount is large, check with i.e. VISA
  - ✓ If amount is even larger, check with issuing bank.

What is the trade-off here?

Concurrency Control: Locking

- To prevent inconsistent updates, locking of resources is a popular mechanism (think of locking data writes on shared resources such as svn or git).

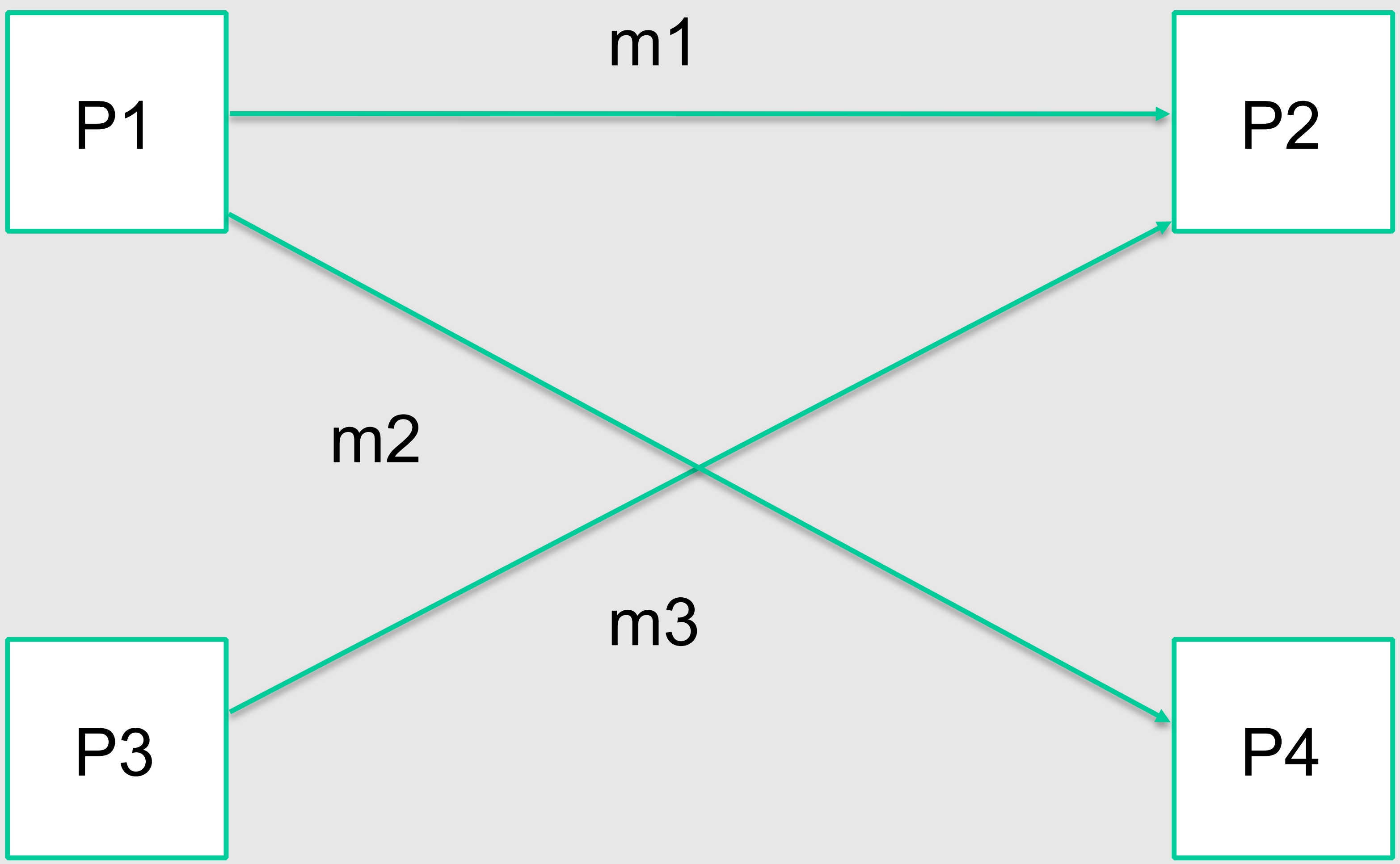
- Another useful mechanism in concurrency control is callback:

  ✓ Server notifies interested clients when there is a change in the security state.

  ✓ For instance: pre-authorisation of credit card fees “locks” money in account. If account is cancelled, merchant is notified.

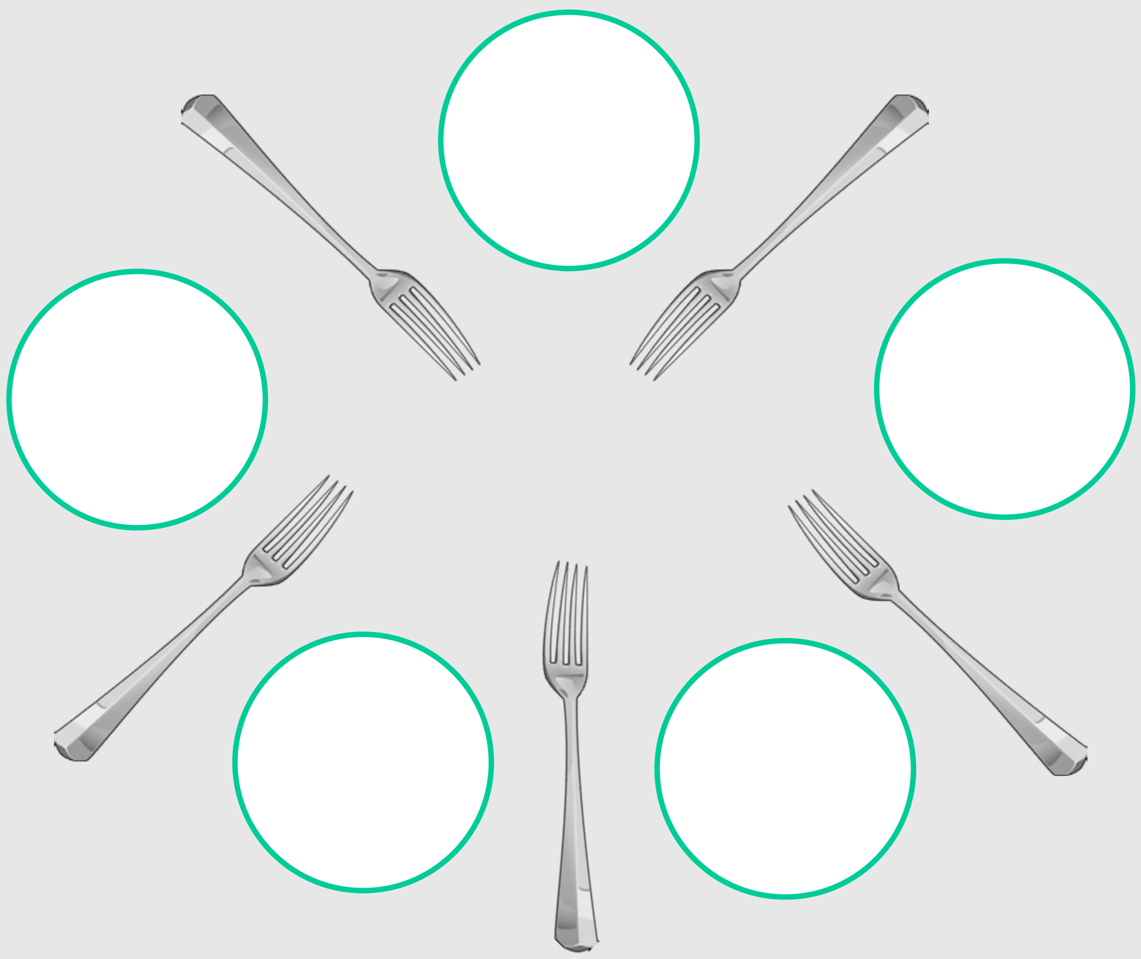
- Proper locking is a complicated science!

Concurrency & Availability



Is a deadlock possible? If so this affects availability.

Concurrency & Availability



- Avoiding deadlocks is hard, think of dining philosophers problem (in practice complexity is much higher).
- Solution by Dijkstra using semaphores (homework).

Concurrency & Availability

- Dining philosophers problem:
  - ✓ Philosophers think deep thoughts, but have simple secular needs.
  - ✓ When hungry, a group of N philosophers will sit around a table with N chopsticks interspersed between them.
  - ✓ Each philosopher enjoys a leisurely meal using the chopsticks on both sides to eat, and 2 sticks are required.
  - ✓ They are exceedingly polite and patient, and each follows the dining protocol:

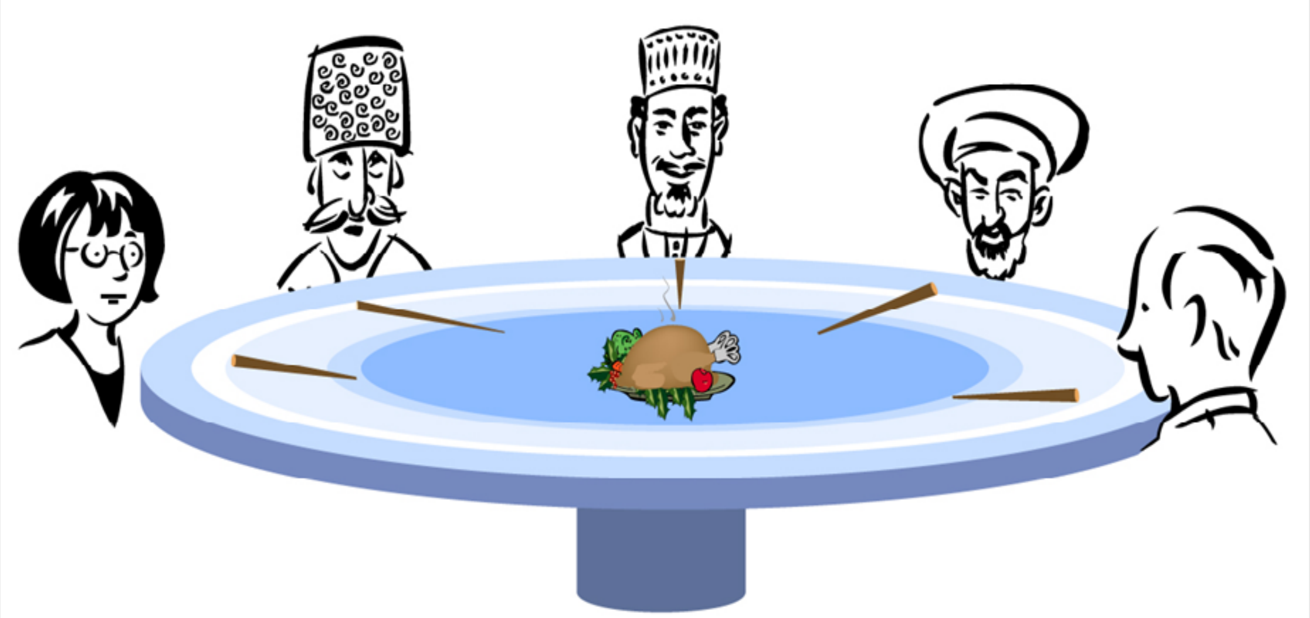


Figure by MIT OpenCourseWare.

Philosophers' dinning protocol:

- **Take (wait for) LEFT stick**
- **Take (wait for) RIGHT stick**
- **EAT until sated**
- **Release both sticks**

Concurrency & Availability

- **Deadlock:** No one can make progress because they are all waiting for an unavailable resource.

- Conditions:
  - ✓ Mutual exclusion – only one process can hold a resource at a given time.
  - ✓ Hold-and-wait – a process holds allocated resources while waiting for others.
  - ✓ No preemption – a resource can not be removed from a process holding it.
  - ✓ Circular wait.

Figure by MIT OpenCourseWare.

- Solution by Dijkstra using semaphores (homework).

Concurrency & Semaphores

- <u>Semaphores</u> for resource allocation:
  - ✓ POOL of K resources.
  - ✓ Many processes, each needs resource for occasional uninterrupted periods.
  - ✓ MUST guarantee that at most K resources are in use at any time.

- Semaphore solution:

In shared memory:

```
semaphore s = K;  /* K resources  */
```

In each process:

```
...
wait(s);     /* Allocate one      */
...          /* use it for a while */
signal(s);   /* return it to pool  */
...
```

Invariant: Semaphore value = number of resources left in pool

Concurrency & Semaphores

- Bounded buffer problem (with a semaphore):

SHARED MEMORY:

```
char buf[N];            /* The buffer */
int in=0, out=0;
semaphore chars=0;
```

PRODUCER:

```
send(char c)
{
    buf[in] = c;
    in = (in+1N;
    signal(chars);
}
```

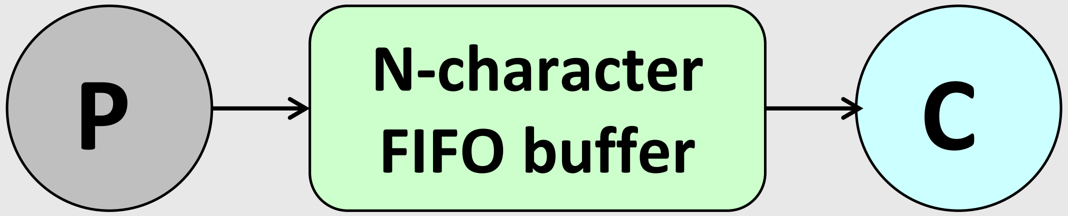
CONSUMER:

```
char rcv()
{   char c;
    wait(chars);
    c = buf[out];
    out = (out+1)%N;
    return c;
}
```

RESOURCE managed by semaphore: Characters in FIFO.

Does it work?

Concurrency & Semaphores



- Flow control problem:

Q: What keeps PRODUCER from putting N+1 characters into the N-character buffer?

A: Nothing. Result: Buffer overflow!

WHAT we've got thus far:

Buffer is not empty when reading.

WHAT we still need:

Buffer is not full when writing.

Concurrency & Semaphores

- Bounded buffer problem (with more semaphores):

SHARED MEMORY:

```
char buf[N];              /* The buffer */
int in=0, out=0;
semaphore chars=0, space=N;
```

PRODUCER:

```
send(char c)
{
   wait(space);
   buf[in] = c;
   in = (in+1)%N;
   signal(chars);
}
```

CONSUMER:

```
char rcv()
{
   char c;
   wait(chars);
   c = buf[out];
   out = (out+1)%N;
   signal(space);
   return c;
}
```

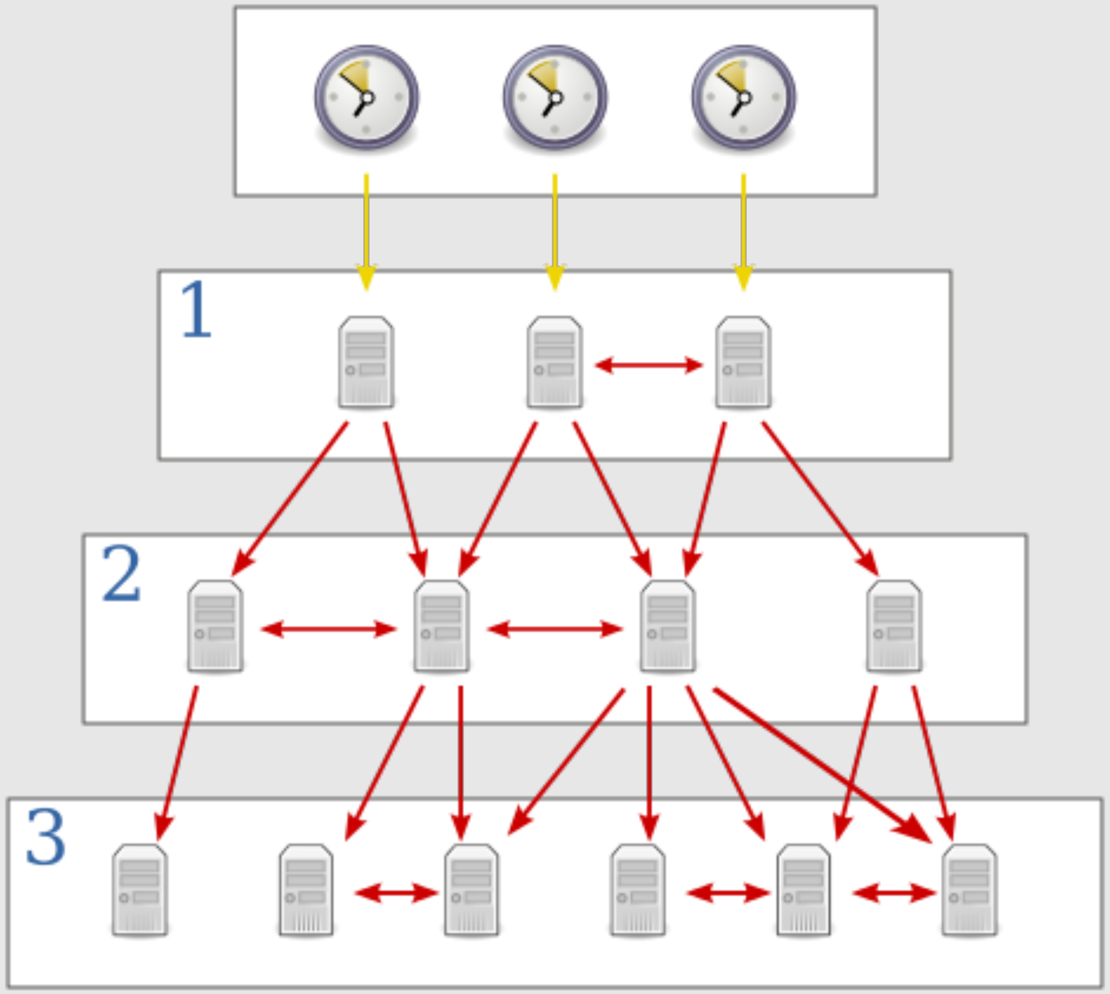
RESOURCEs managed by semaphores: Characters in FIFO, Spaces in FIFO

Secure Time

- Why is time important? How to reliably get current time?
  - ✓ If we rely on network servers, attackers might tamper with time.
  - ✓ This can have severe consequences (as we will discuss in crypto protocols).
- Even using GPS or Radio Clocks, motivated attackers can tamper with those signals.
- In practice however the Network Time Protocol (NTP, port 123) is dependable for most applications.

Secure Time

- **Network Time Protocol (NTP):**



Atomic clock

Broadcast to different layers of servers

Attacks on secure time service

Majority clock voting &
authentication of time server

- NTP can usually maintain time to within tens of milliseconds over the public Internet.

Fault Tolerance

- Failure tolerance/recovery is a very important aspect of security engineering! (Attacks/Faults eventually will happen).

- Research is typically focused on confidentiality/integrity, however in practice much of the budget goes to resilience against failure.
  - ✓ Think of any service you know (banks, social networks, shops) → *Availability* is key for operation.
  - ✓ Twitter downtime estimated cost is 25 million per minute.

    http://www.cnet.com/news/the-cost-of-twitter-downtime/

- A **fault** may cause an **error** (incorrect state), this may lead to a **failure** (deviation from system's specified behaviour).

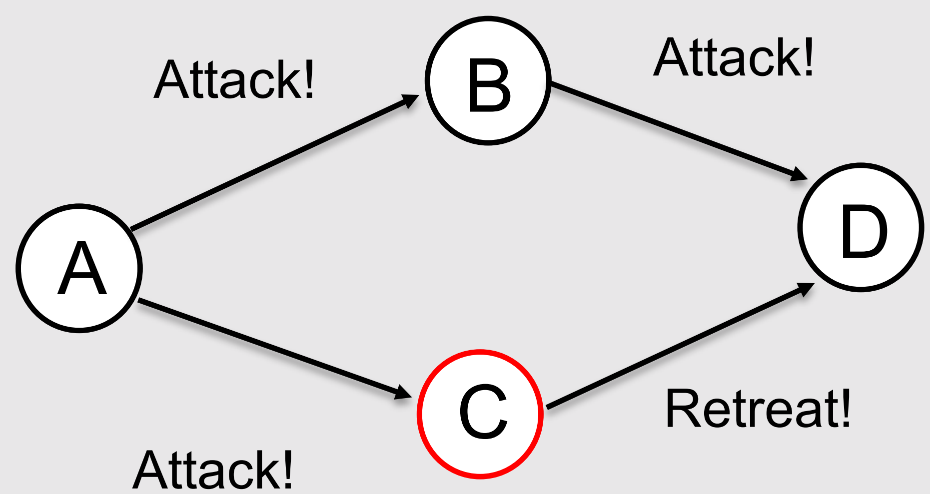
- How long does it take for a system to fail after an error occurs, how long does it take to recover?

Byzantine Failure

- Assume there are *n* generals defending Byzantium, *t* of them are adversarial (have been bribed).

  ✓ They communicate with each other using messages. Traitors will try to confuse loyal generals.

- What is the maximum number of traitors that can be tolerated?



n >= 3t + 1 (Lamport et al.)

Proof ? (homework)

Fault Tolerance

- How to cope with failures?
- Two basic strategies:
  - ✓ Redundancy
    - ❖ Goal: Preserve integrity and availability of data by having multiple copies of it and checking them against each other.
    - ❖ Examples: Backups, duplicated systems, increase number of sensors.
  - ✓ Fail-stop
    - ❖ Goal: Detect failures and stop processing.
    - ❖ Examples: Monitors checking invariants of the system (like checksums, balance checks).

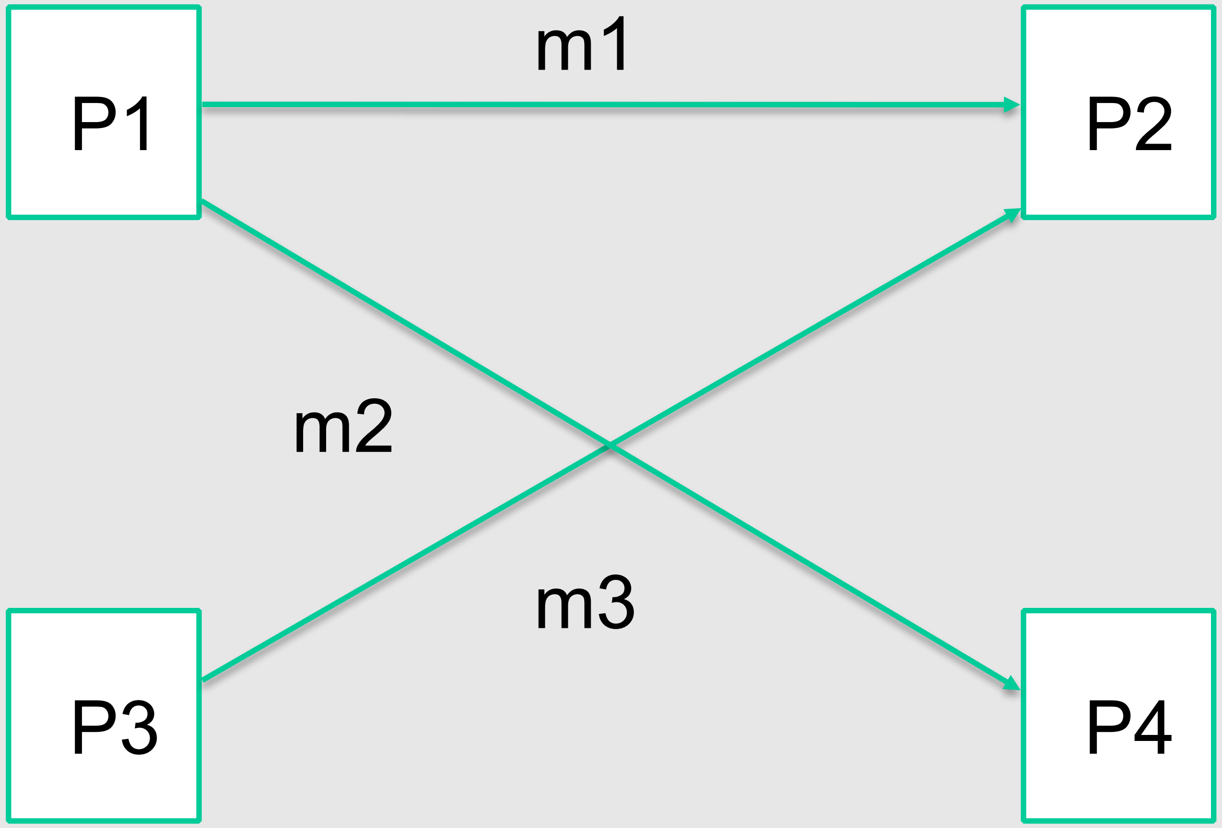
Fault Tolerance

- How to cope with failures?
- Two basic strategies:
  - ✓ Redundancy
    - ❖ Goal: Preserve integrity and availability of data by having multiple copies of it and checking them against each other.
    - ❖ Examples: Backups, duplicated systems, increase number of sensors.
    - ❖ **Problem:**: Higher risk of confidentiality attacks!
  - ✓ Fail-stop
    - ❖ Goal: Detect failures and stop processing.
    - ❖ Examples: Monitors checking invariants of the system (like checksums, balance checks).
    - ❖ **Problem:**: Higher risk of availability attacks!

Denial of Service

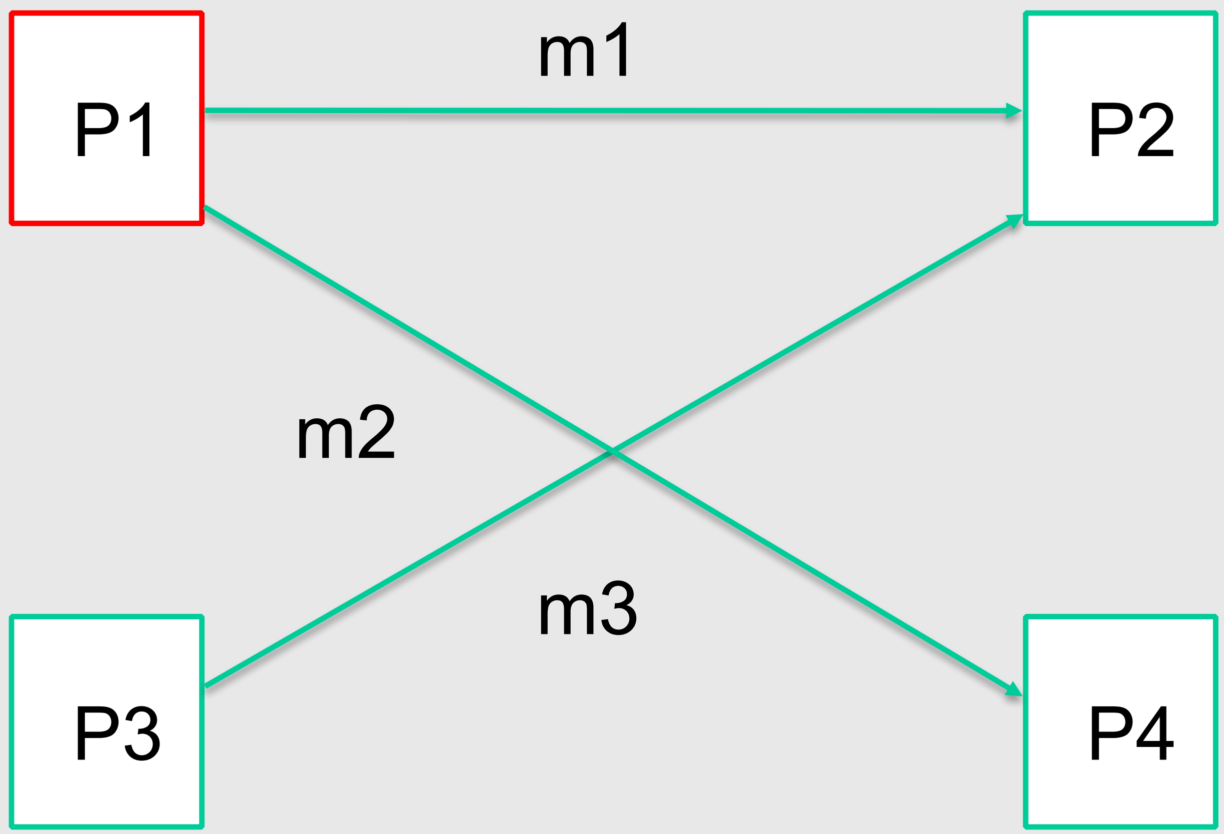
- Components of a system might be forced to fail by malicious adversaries with different purposes:
  - ✓ Take a service down.
  - ✓ Example?.
- Nowadays even harder with Distributed Denial of Service attacks (DDoS).
  - 僵尸网络 ✓ A botnet of infected machines can be used to flood a system.
  - ✓ How do you know which machines to block and which not to?
  - ✓ Need for extreme replication!

Safety vs. Security



- People building safety critical systems have been concerned with the following problem for decades:
  - ✓ Can a distributed system reach a safety-harming state?

Safety vs. Security



- People building safety critical systems have been concerned with the following problem for decades:
  - ✓ Can a distributed system reach a safety-harming state?
- In security main philosophical difference: attacker is unpredictable and if there is a vulnerability, he will exploit it.

Guarantees?

- There is a large body of research that tackles this problem from a verification perspective.
  - ✓ Use formal models of systems (Like the automata we saw in the last lecture).
  - ✓ Formalize desirable properties as mathematical properties of states and traces.
  - ✓ Use theorem proving or model-checking to prove/disprove properties on concrete models.
- There are promising results and applications to safety critical systems such as space shuttles, trains, planes.
- However reasoning about security is typically harder and cost benefit relation is not clear.

Key Points

- Availability issues in distributed systems
- Concurrency
  - ✓ Race conditions → atomicity
  - ✓ Deadlocks → semaphores
- Fault tolerance
  - ✓ Redundancy → higher risk of confidentiality attacks
  - ✓ Fail-stop → higher risk of availability attacks

Exercises & Reading

- Classwork (Exercise Sheet 4): due on Fri Oct 5, 10:00 PM
- Homework (Exercise Sheet 4): due on Fri Oct 12, 6:59 PM
- Reading: RA [Ch6]
- Mid-term exam (Week 6): Fri 19 Oct, 7:30 PM (covering Part I Foundations: Week 1 – Week 4)

End of Slides for Week 4