Deadlock

4010-441
Principles of Concurrent System Design

Topic Outline

- Deadlocks
  - Desired access properties for shared mutable resources
  - Classic deadlock example: Dining Philosophers
  - Root causes and four necessary and sufficient conditions
  - Deadlock prevention / avoidance / detect + repair

When using shared, mutable resources, there are several access properties your system should exhibit.

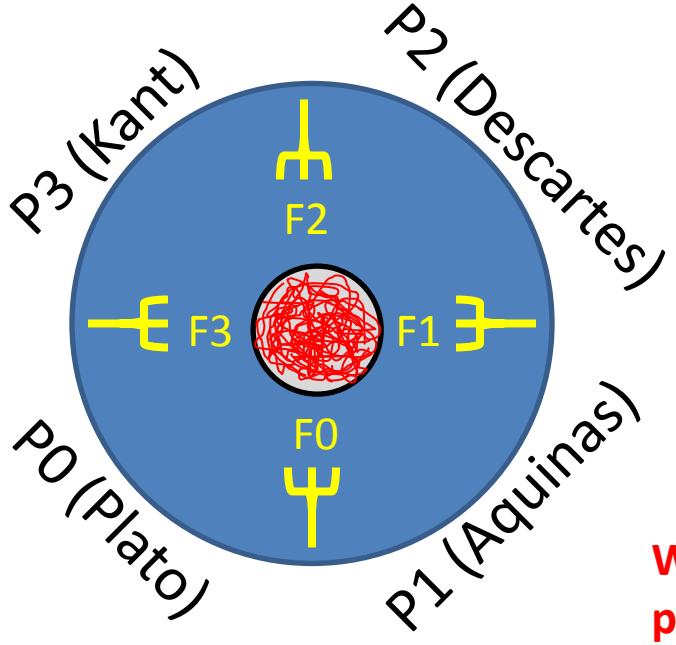
- A **shared, mutable resource** (SMR) could be a shared mutable variable, or a device such as a communication channel, disk, or printer.

What properties do you want your system to exhibit with respect to access to an SMR?

- Safety (job #1):
  Mutually exclusive access to shared, mutable resource (SMR)
- Liveness 1:
  If threads are trying to access an SMR, one eventually does.
- Liveness 2:
  A thread holding an SMR eventually releases it.
- Fairness (no starvation):
  If a thread is trying to access an SMR, it eventually gains access.

The classic Dining Philosophers can deadlock and leave the philosophers hungry.

- Informally – a set of threads blocked with no possibility of progress.
- Formally – a set of threads, each holding an SMR needed by another thread in the set and waiting to acquire a resource which is already held
- Classic example: Dining Philosophers



What is a path to deadlock?

- Naïve Approach
  - Get right fork
  - Get left fork
  - Eat

What conditions exist that permit this deadlock?

There are **four necessary and sufficient conditions** for deadlock to be possible.

- *Necessary* means all must hold for *deadlock to be possible*.
- *Sufficient* means if all hold *deadlock is possible*.
- The four necessary and sufficient conditions for deadlock to be possible are
  - Exclusive use of resources
  - No preemption of resource hold
  - Serial acquisition of resources
  - Cyclic hold-and-wait graph
- Having these four conditions guarantees that deadlock is possible. It does not guarantee that it will happen.
  - Do you want to trust your system with "it may not happen"?

How could we remove each of these conditions in the Dining Philosophers?

Observations

- Deadlock can occur with both individual and pooled resources.
- Goal is to design deadlock out of the system
  - Eliminate one of the four conditions
  - Use allocation methods, such as, Bankers Algorithm, that will not allocate into an unsafe state
- Detect and recover:
  - Detection – periodically scan allocation graph for deadlocks
  - Recover – kill a thread
- Use a different concurrency mechanism not prone to deadlock
  - Software Transaction Memory later in the term