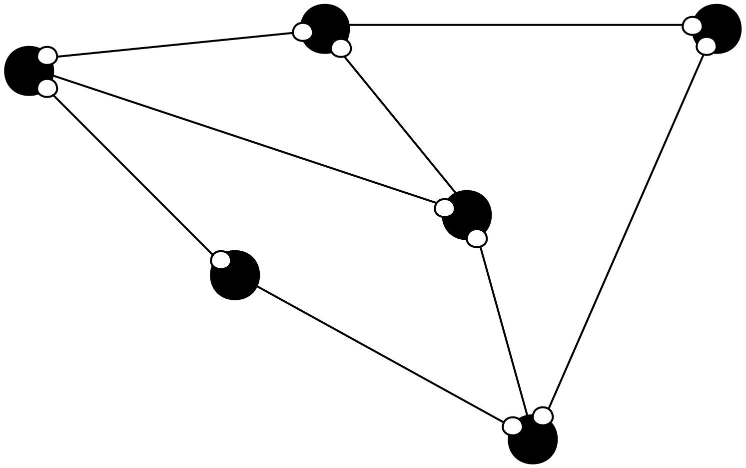
Using Diposets to Model Concurrent Systems

John S. Davis II
IBM T.J. Watson Research Center

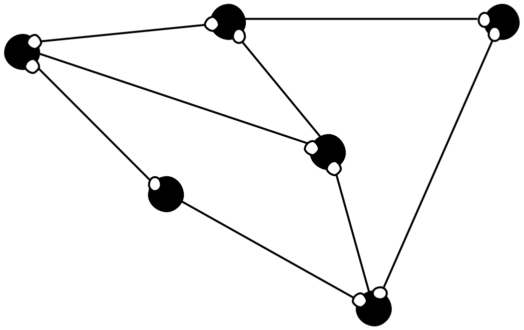
Concurrent System

- A concurrent system is a network of communicating components.



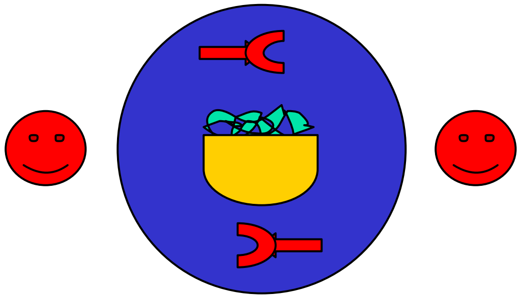
Design Is Difficult

- Concurrent systems are very difficult to design.
    - Component relationships must be managed.
    - Improper relationship management can lead to:
        - Interference
        - Deadlock
        - Indeterminacy



The Canonical Concurrent System

- Dining Philosophers
  - N philosophers share N forks.
  - Each philosopher uses two forks at a time to eat spaghetti.
- The Dining Philosophers can deadlock!
- Components Include:
  - Philosophers
  - Forks
- Component Relationships?



DiningPhilosophers Class

```
public class DiningPhilosophers {
        public void main() {
                    philo1.start();
                    philo2.start();
                    while( philosophersStillAlive() ) {
                        wait();
                    }
        }

        public synchronized boolean philosophersStillAlive() {
                    return ( philoCnt != 0 );
        }
        public synchronized void reducePhilosopherCount() {
                    philosopherCnt--;
        }

        Philosopher philo1 = new Philosopher(fork1, fork2);
        Philosopher philo2 = new Philosopher(fork1, fork2);
        private Object fork1 = new Object();
        private Object fork2 = new Object();
        private philosopherCnt = 2;
}
```

- Consider the software component relationships…

Philosopher Class

```
public class Philosopher extends Thread {
        Philosopher(Object lk1, Object lk2) {
                fork1 = lk1; fork2 = lk2;
        }

        public void run() {
                if ( random() < 0.5 ) {
                    synchronized( fork1 ) {
                        synchronized( fork2 ) {
                                getSpaghetti();
                                eat();
                        }
                    }
                } else {
                    synchronized( fork2 ) {
                        synchronized( fork1 ) {
                                eat();
                                burp();
                        }
                    }
                }
        }
        private Object fork1, fork2;
}
```

- Consider the software component relationships…

Two Especially Important Relationships

1) The Order Relationship

- Relative timing of events
  - What is the order of actions or method calls?

```
if( random() < 0.5 ) {
    synchronized(fork1) {
        synchronized(fork2) {
            getSpaghetti();
            eat();
        }
    }
}
```

getSpaghetti() first,
then eat().

Two Especially Important Relationships

2) The Containment Relationship

- Hierarchical organization of events
  - How are code blocks nested?

```
} else {
    synchronized(fork2) {
        synchronized(fork1) {
            eat();
            burp();
        }
    }
}
```

The synchronization of fork2 contains the synchronization of fork1 which contains the eat() and burp() methods.

The Challenge

- Challenges of designing concurrent systems
  - Humans think sequentially
  - Simultaneity is a difficult concept independent of a global clock
- Software is invisible[1]
  - Concurrent software is geometrically invisible!

[1] "No Silver Bullet," Fred Brooks Jr., 1987

Design Tools Are Needed

- Good concurrent system modeling and design tools are needed.
- A good tool should be
  - Unambiguous
  - General
  - Graphical
  - Formal

Available Tools Are Insufficient

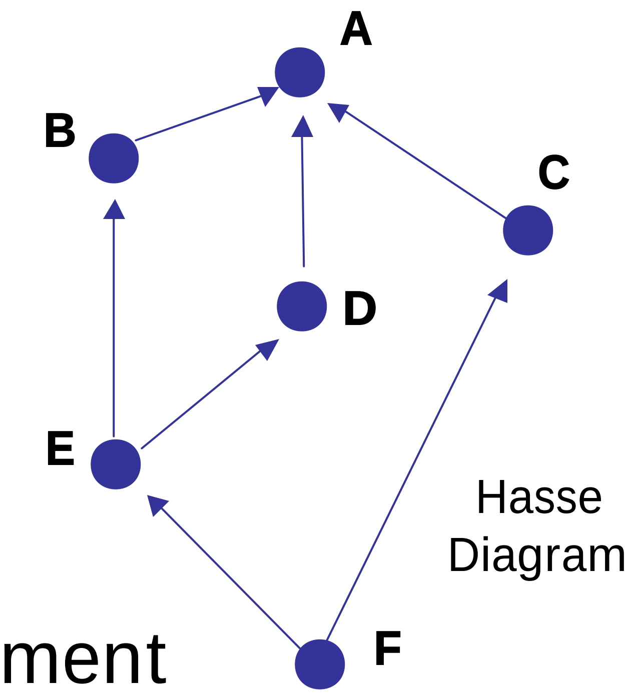
- Software Patterns
- UML
- Call Graphs
- Petri Nets
- Temporal Logic
- *What about modeling order?*
  - Partially Ordered Sets

Partially Ordered Sets (Posets)

- Poset = (X, R)
  - X - Ground Set
  - R - Relation on X
    - Reflexive: $x \le x$
    - Anti-symmetric: $x \le y$, $y \le x$ implies $x = y$
    - Transitive: $x \le y$, $y \le z$ implies $x \le z$
  - $x \le y$ or $y \le x$ means that x is comparable to y. Otherwise, x and y are incomparable.

Poset Hasse Diagrams

- Cover Element of x
  - Minimal elements of the upset of x.
- Example
  - A, B & D are greater than F. C & E cover F.
- Only one relation
  - Not order *and* containment



Hasse Diagram

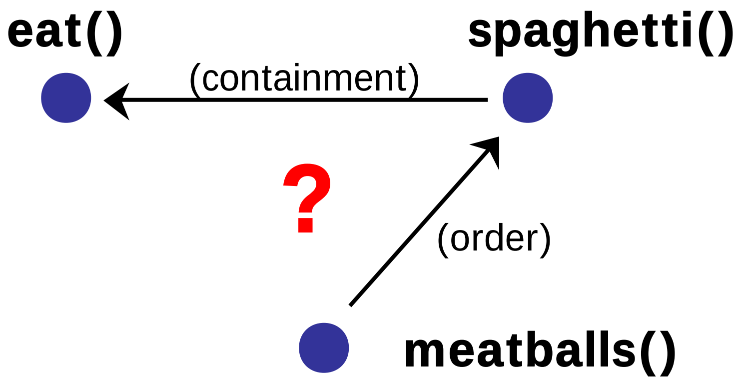
Software w.r.t. Posets

- The simplest software systems can not be modeled by posets.
  - Two relations are needed:

    1) Order

    2) Containment

```
eat() {
    spaghetti();
    meatballs();
}
```



Paired Directed Graphs

- Graph, $G = (V, E)$
  - V – Set of nodes
  - E – Set of edges (pairs of nodes)
- Paired Directed Graph, G_{PD}
  - $G_{PD} = (V, E)$ s.t. $E = \{E_O \text{ union } E_C\}$
  - E_O – Represents Order Relation
  - E_C – Represents Containment Relation

Plenty of Freedom

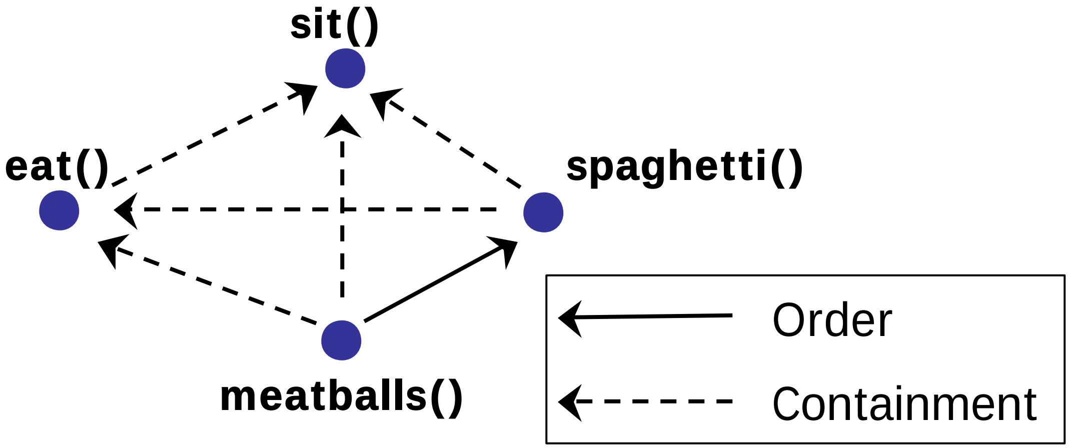
- Paired Directed Graphs (G_{PD}s)
    - Can represent a wide variety of software relationships:
        - Order/Containment Relationships
        - Static/Dynamic Instances

Too Much Freedom

- G_{PD}s have too little structure.
  - Freedom begets complexity.
  - Reduce complexity via implied relationships.

Complexity:

```
sit() {
    eat() {
        spaghetti();
        meatballs();
    }
}
```



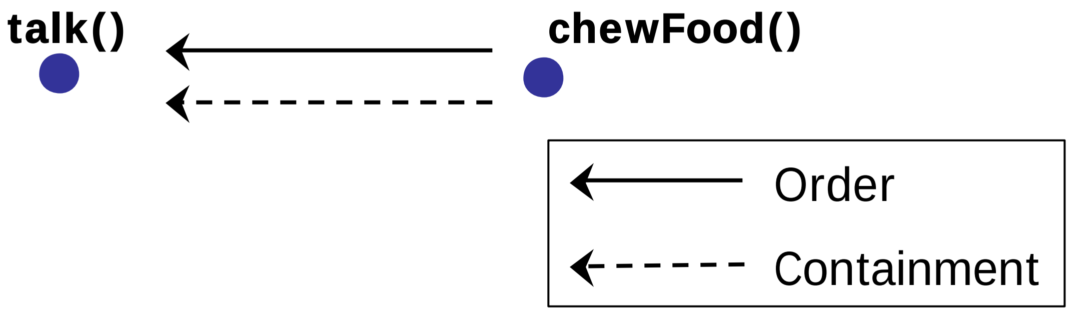
Too Much Freedom

- G_{PD}s have too little structure.
  - Freedom enables weirdness.
  - Avoid weirdness by appropriate constraints.

Weirdness:

What kind of Code:
talk() & chewFood()

?

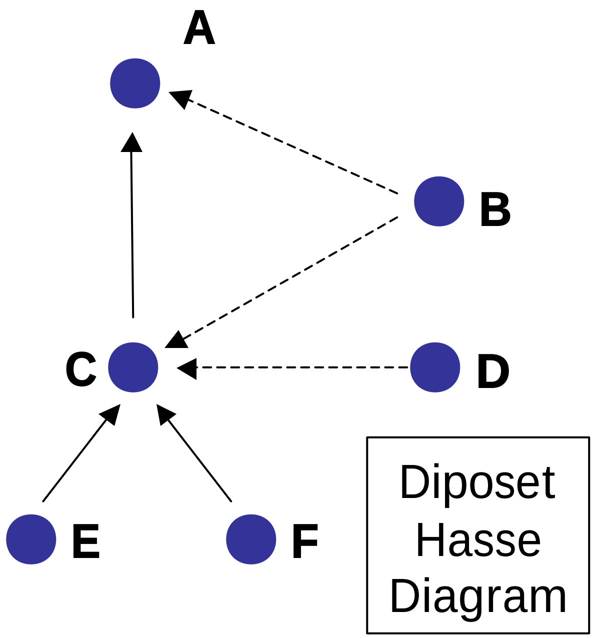


Diposets

- Diposet = (X, R_O, R_C)
  - X - Ground Set
  - R_O - Order Relation on X
  - R_C - Containment Relation on X
  - If $(x,y) \in R_O$ then $(x,y), (y,x) \notin R_C$
  - Order
    - Comparable/Incomparable
  - Containment
    - Inclusive/Mutually Non-inclusive

Diposet Hasse Diagrams

- Must show two relations[2]
  - Solid arrow
    - Order Relation
  - Dashed arrow
    - Containment Relation
- Examples:
  - B is contained in A, C
  - C is preceded by A

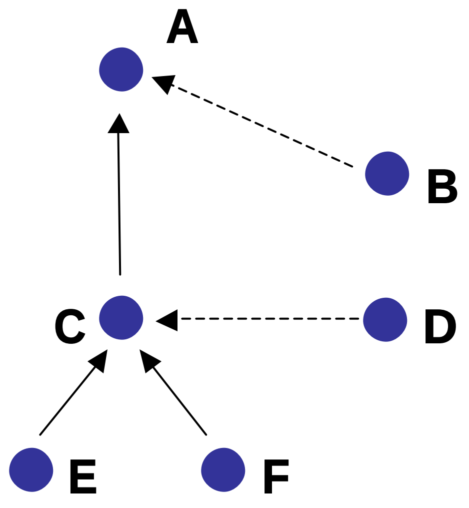


Diposet Hasse Diagram

[2] Alternative arrow representations are also possible.

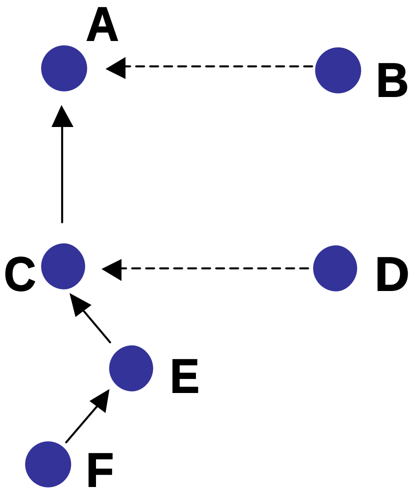
Nested Diposets

- A nested diposet is a diposet in which:
  1. x,y mutually non-inclusive implies that the contents of x,y are mutually non-inclusive.
  2. x,y ordered implies that the contents of x (y) are ordered with respect to y (x).



Sequential Nested Diposets

- A sequential nested diposet is a nested diposet in which:
  1. There exists a maximum container.
  2. If x,y have a common container cover, then x,y must be ordered w.r.t. each other.
- Equivalent to a thread.

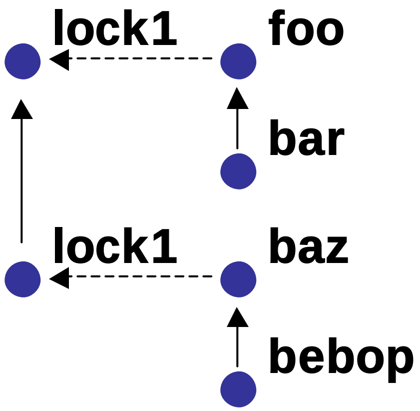


Concurrent Programming

- Represent dynamic method call instances as events.
- Bold Conjectures:
  - Concurrent Program =
    - Set of paired directed graphs.
  - Deadlock Free Concurrent Program =
    - Set of nested diposets.
  - Thread =
    - A sequential nested diposet.

Safety and Synchronization

- Synchronization locks can be represented by containment.
- Distinct invocations of a given lock must be ordered.



Liveness and Deadlock

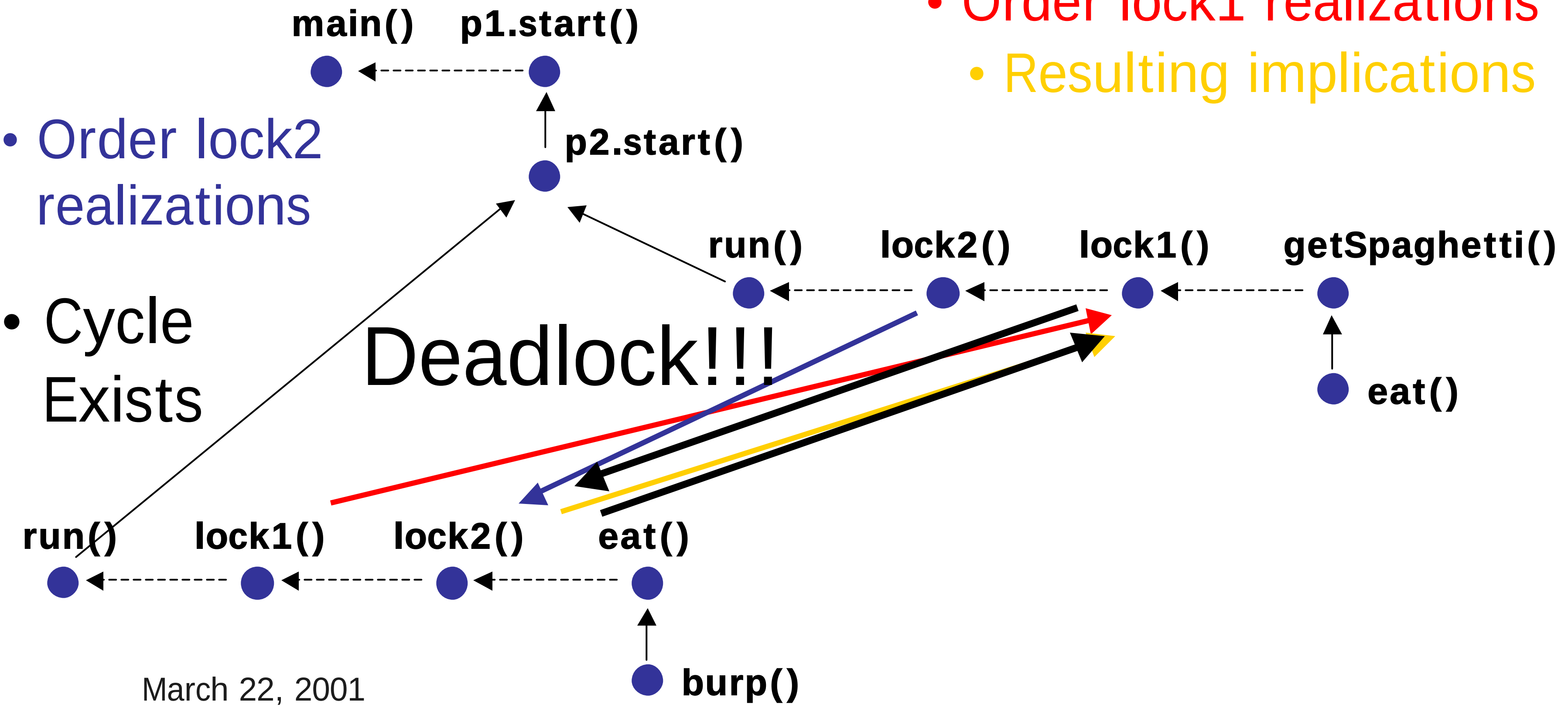
- Acyclic Diposet Theorem
  - A diposet can not contain an order or containment cycle of length 2 or more.
- Cycles Imply Deadlock
  - A paired directed graph is deadlock prone iff it contains a cycle.
  - If a program can be represented by a nested diposet, it will not exhibit deadlock.

Dining Philosophers Revisited

- Order lock1 realizations
  - Resulting implications

- Order lock2 realizations

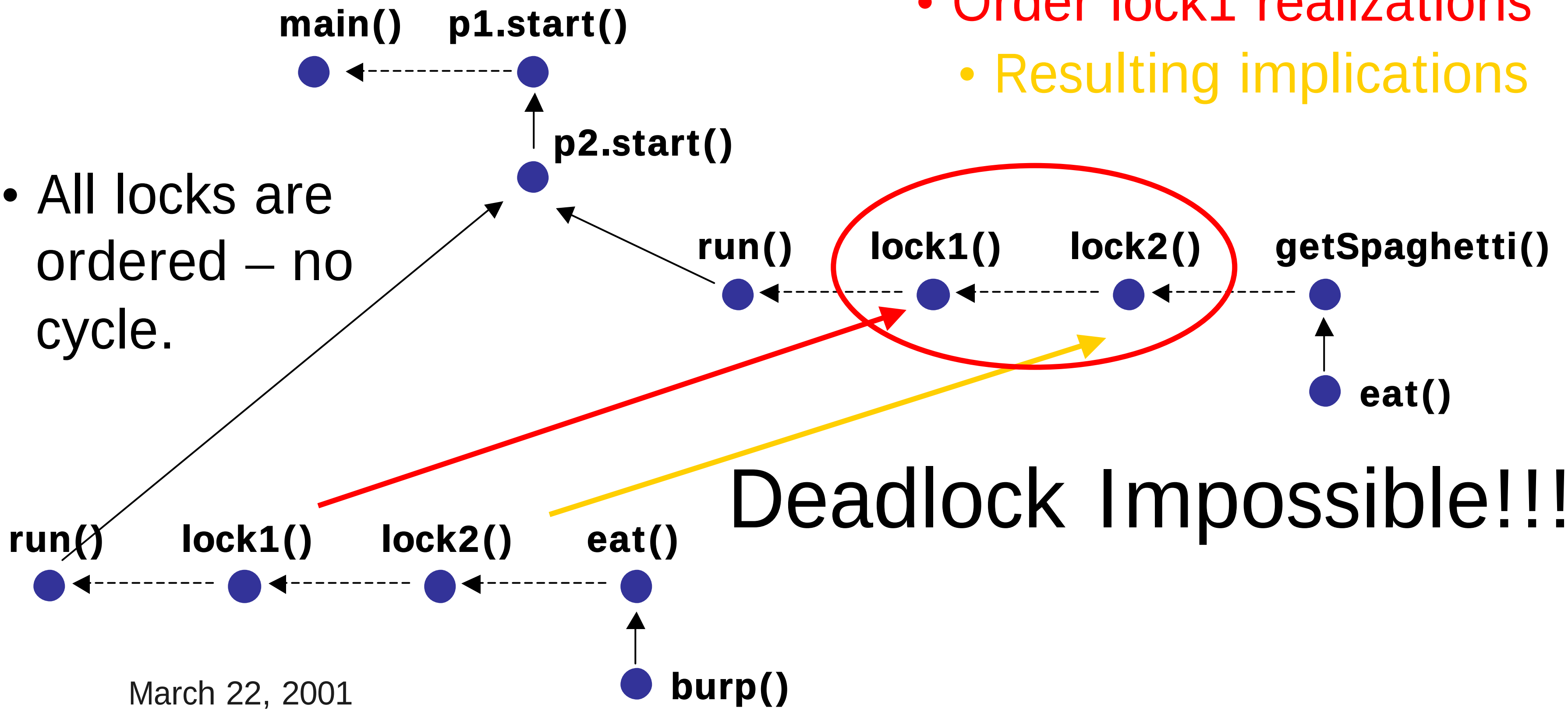
- Cycle Exists



Dining Philosophers Without Deadlock

- Order lock1 realizations
  - Resulting implications

- All locks are ordered – no cycle.

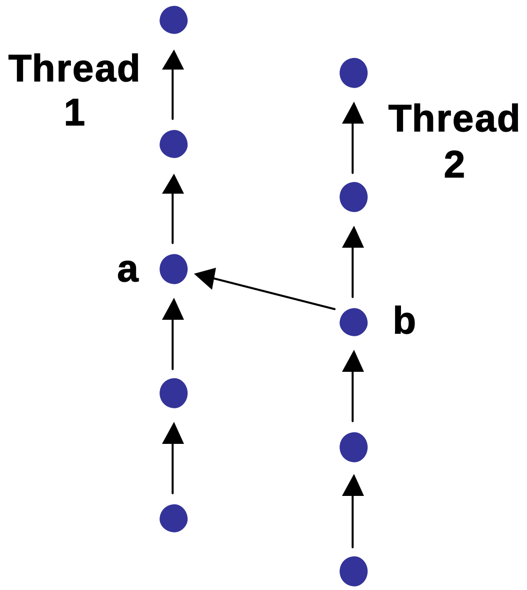


main()
p1.start()
p2.start()
run()
lock1()
lock2()
getSpaghetti()
eat()
run()
lock1()
lock2()
eat()
burp()

Deadlock Impossible!!!

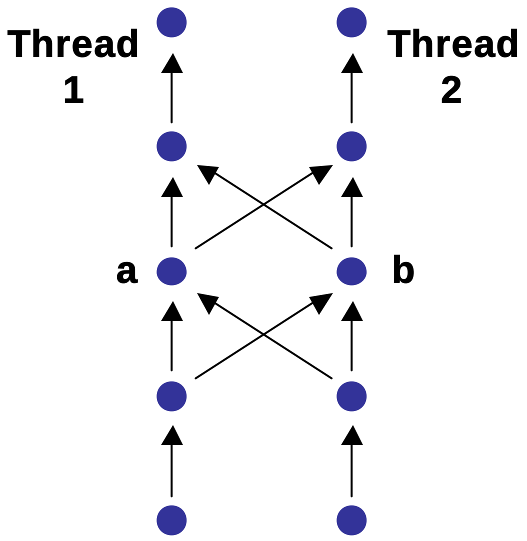
Communication Semantics

- Asynchronous Message Passing
  - Asynchronous communication between events a and b is represented. Polarity is implied (b receives communication from a).



Communication Semantics

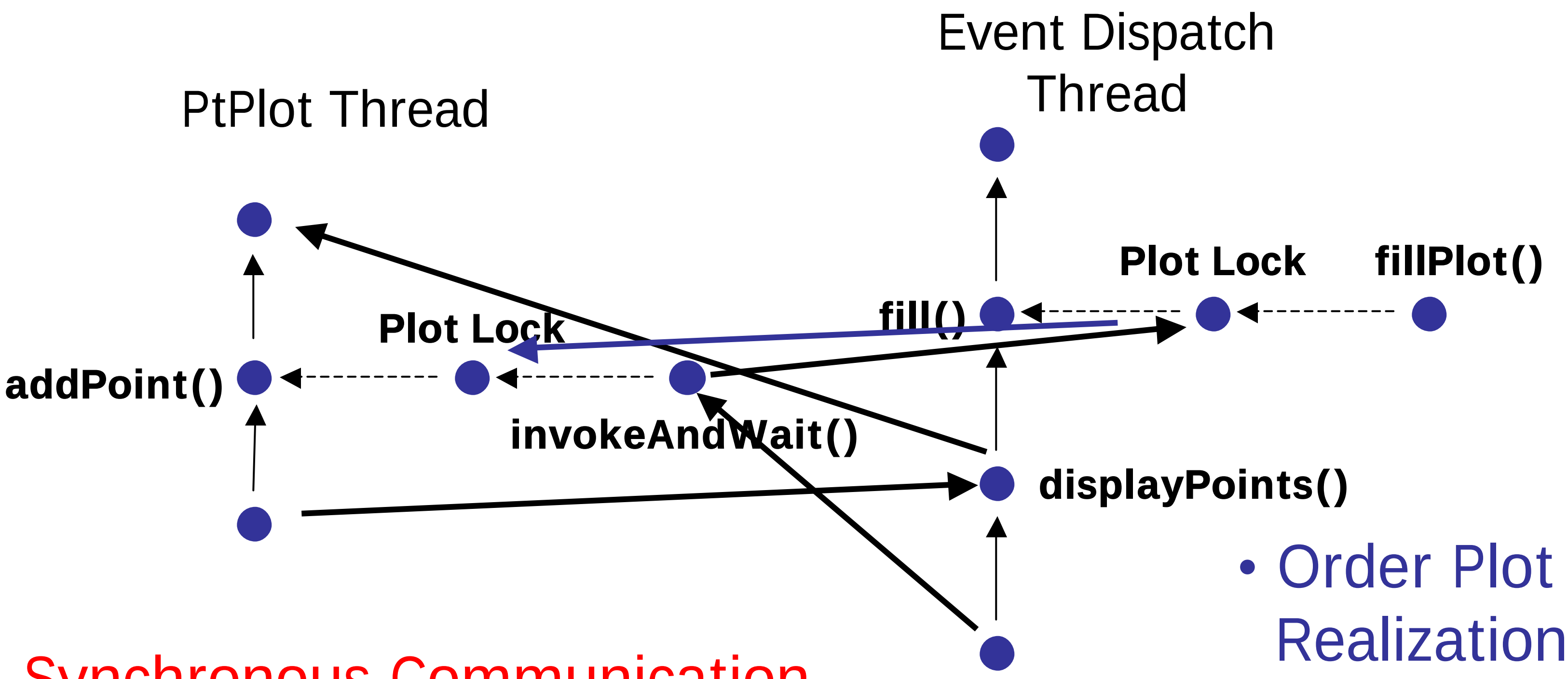
- Synchronous Message Passing
  - Synchronous communication between events a and b is represented. Polarity is not implied.



PtPlot Example

- PtPlot
  - Graphical display package by Edward A. Lee and Christopher Hylands
- High Level Architecture
  - Utilizes the Java Swing Package that uses an Event Dispatch Thread for GUI events.
  - A long running PtPlot Thread communicates with the Event Dispatch Thread.
- Development required intense debugging

Thread Interaction in PtPlot

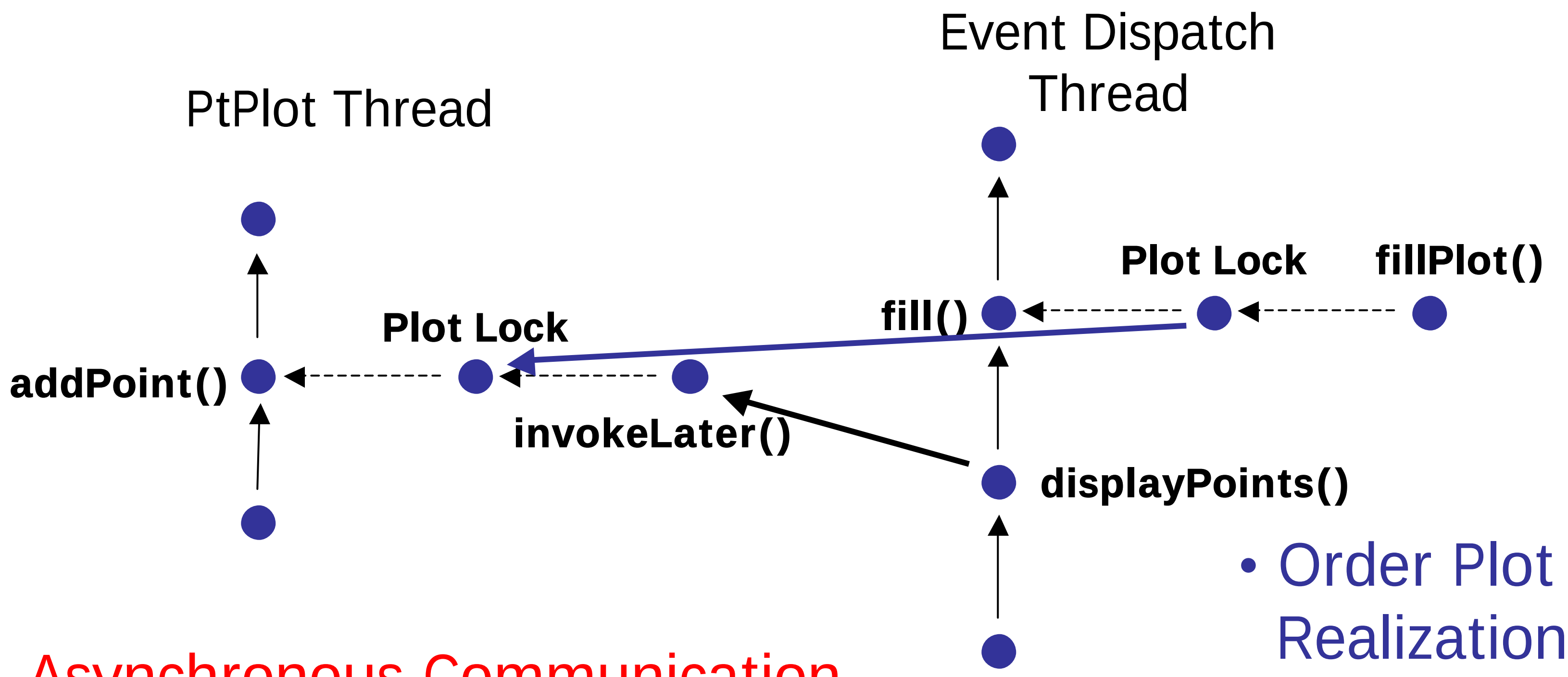


Synchronous Communication

Cycle – Deadlock Possible!!!

- Order Plot Lock Realizations
- Communication Order Constraints

Thread Interaction in PtPlot



- Order Plot Lock Realizations
- Communication Order Constraints

Asynchronous Communication

No Cycle – Deadlock Impossible!!!

Conclusion

- Diposets offer an intuitive but formal framework
- Paired Directed Graphs serve as a foundation for general concurrent semantics.
- Future Work
  - Develop an operational semantics.