Avoiding Coordination
Announcements
Office Hours

• No office hours on 11/15: I am not here.
Office Hours

- No office hours on 11/15: I am not here.

- No office hours on 11/22: Everyone should be out having 🥧 and 🦃.
Office Hours

• No office hours on 11/15: I am not here.

• No office hours on 11/22: Everyone should be out having 🍰 and 🦃.

• No office hours on 11/29: I am not here.
Office Hours

- No office hours on 11/15: I am not here.
- No office hours on 11/22: Everyone should be out having 🍰 and 🦃.
- No office hours on 11/29: I am not here.
- E-mail me if you want to meet/need help.
Office Hours

• No office hours on 11/15: I am not here.

• No office hours on 11/22: Everyone should be out having 🍩 and 🦃.

• No office hours on 11/29: I am not here.

• E-mail me if you want to meet/need help.

  • Probably going to be over Skype or some such medium.
Office Hours

• No office hours on 11/15: I am not here.

• No office hours on 11/22: Everyone should be out having 🥧 and 🦃.

• No office hours on 11/29: I am not here.

• E-mail me if you want to meet/need help.

  • Probably going to be over Skype or some such medium.

  • Or in person on 11/26 or 11/27.
Final Project
Final Project

• Hopefully everyone has started...
Final Project

• Hopefully everyone has started...

• ... at least thinking about the project.
Final Project

• Hopefully everyone has started...
  • ... at least thinking about the project.
• Remember: no extensions or late days -- must receive final report on Dec 12.
Onto Science...
So Far...

- Looked at mechanisms to achieve consensus, get machines to work together.
So Far...

- Looked at mechanisms to achieve consensus, get machines to work together.
- A few different algorithms, a few different failure conditions.
So Far...

- Looked at mechanisms to achieve consensus, get machines to work together.
- A few different algorithms, a few different failure conditions.
- Powerful primitive: can replicate any deterministic state machine.
So Far...

- Looked at mechanisms to achieve consensus, get machines to work together.
  - A few different algorithms, a few different failure conditions.
- Powerful primitive: can replicate any deterministic state machine.
  - Don't even need to implement or understand consensus to do this.
So Far...

• Looked at mechanisms to achieve consensus, get machines to work together.
  • A few different algorithms, a few different failure conditions.
  • Powerful primitive: can replicate any deterministic state machine.
  • Don't even need to implement or understand consensus to do this.
  • Powerful abstraction: replicate most programs, get fault tolerance.
So Far...
So Far...

• But everything is not rosy:
So Far...

• But everything is not rosy:
  • Configure and initialize system for correctness.
So Far...

But everything is not rosy:

- Configure and initialize system for correctness.
- Need to communicate and wait before responding to any request.
So Far...

• But everything is not rosy:
  • Configure and initialize system for correctness.
  • Need to communicate and wait before responding to any request.
  • Can we do better?
Yes?
Self Stabilizing Algorithms
Setting

- Treat program as transition machine.
Setting

• Treat program as transition machine.
• Where some states are good.
Setting

• Treat program as transition machine.
• Where some states are *good*.
• All transitions from good states go to good.
Treat program as transition machine.
Where some states are good.
All transitions from good states go to good.
Self-stabilization:
From any state arrive at good state.
In bounded steps.
Setting

- Treat program as transition machine.
- Where some states are good.
- All transitions from good states go to good.
- Self-stabilization:
  - From any state arrive at good state.
  - In bounded steps.
Djikstra's Example

- Mutual exclusion: only one process gets to write (or compute) at a time.
Dijkstra's Example

- Mutual exclusion: only one process gets to write (or compute) at a time.

At process 0

do {
    if (x_4 == x_0) {
        x_0 = (x_0 + 1) % 5;
    }
} while (true);

At process n

do {
    if (x_n != x_{n-1}) {
        x_n = x_{n-1};
    }
} while (true);
Dijkstra's Example

• When and why can this provide mutual exclusion?

At process 0
do {
    if (x₄ == x₀) {
        x₀ = (x₀ + 1) % 5;
    }
} while (true);

At process n
do {
    if (xₙ != xₙ₋₁) {
        xₙ = xₙ₋₁;
    }
} while (true);
Dijkstra's Example

At process 0
do {
    if (x4 == x0) {
        x0 = (x0 + 1) % 5;
    }
} while (true);

At process n
do {
    if (x_n != x_{n-1}) {
        x_n = x_{n-1};
    }
} while (true);
Djikstra's Example

At process 0
do {
    if (x_4 == x_0) {
        x_0 = (x_0 + 1) % 5;
    }
} while (true);

At process n
do {
    if (x_n != x_{n-1}) {
        x_n = x_{n-1};
    }
} while (true);
Dijkstra's Example

At process 0
    do {
        if (x_4 == x_0) {
            x_0 = (x_0 + 1) % 5;
        }
    } while (true);

At process n
    do {
        if (x_n != x_{n-1}) {
            x_n = x_{n-1};
        }
    } while (true);
Dijkstra's Example

At process 0
\[
\text{do \{ \\
  \text{if (} x_4 == x_0 \text{) \{ \\
      x_0 = (x_0 + 1) \mod 5; \\
  \}} \\
\} \text{ while (true);} \\
\]

At process n
\[
\text{do \{ \\
  \text{if (} x_n \neq x_{n-1} \text{) \{ \\
      x_n = x_{n-1}; \\
  \}} \\
\} \text{ while (true);} \\
\]
At process 0
   do {
       if (x₄ == x₀) {
           x₀ = (x₀ + 1) % 5;
       }
   } while (true);

At process n
   do {
       if (xₙ != xₙ₋₁) {
           xₙ = xₙ₋₁;
       }
   } while (true);
Dijkstra's Example

What happens with random initial values?

At process 0
```
do {
    if (x_4 == x_0) {
        x_0 = (x_0 + 1) % 5;
    }
} while (true);
```

At process n
```
do {
    if (x_n != x_{n-1}) {
        x_n = x_{n-1};
    }
} while (true);
```
Dijkstra's Example

At process 0
    do {
        if (x₄ == x₀) {
            x₀ = (x₀ + 1) % 5;
        }
    } while (true);
At process n
    do {
        if (xₙ != xₙ₋₁) {
            xₙ = xₙ₋₁;
        }
    } while (true);
Djikstra's Example

- Assuming a fair execution.

At process 0
    do {
        if (x₄ == x₀) {
            x₀ = (x₀ + 1) % 5;
        }
    } while (true);

At process n
    do {
        if (xₙ != xₙ₋₁) {
            xₙ = xₙ₋₁;
        }
    } while (true);
Djikstra's Example

- Assuming a fair execution.
- **Lemma 1**: At least every n rounds $x_0$ changes its value.

At process 0

do {
    if ($x_4 == x_0$) {
        $x_0 = (x_0 + 1) \% 5$;
    }
} while (true);

At process $n$

do {
    if ($x_n != x_{n-1}$) {
        $x_n = x_{n-1}$;
    }
} while (true);
Dijkstra's Example

- Assuming a fair execution.

- **Lemma 1**: At least every n rounds $x_0$ changes its value.

- Round here is every processor got a chance to run.
Djikstra's Example

• Assuming a fair execution.
• **Lemma 1**: At least every $n$ rounds $x_0$ changes its value.
  • Round here is every processor got a chance to run.
• Why?
Djikstra's Example

- Assuming a fair execution.
- **Lemma 1**: At least every n rounds $x_0$ changes its value.
- **Lemma 2**: There is some value $c$ in 0..(n+1) s.t. $x_i \neq c \ \forall i$
Dijkstra's Example

- Assuming a fair execution.

- **Lemma 1**: At least every $n$ rounds $x_0$ changes its value.

- **Lemma 2**: There is some value $c$ in $0..(n+1)$ s.t. $x_i \neq c \ \forall i$

- Why?
Dijkstra's Example

- Assuming a fair execution.
- **Lemma 1**: At least every $n$ rounds $x_0$ changes its value.
- **Lemma 2**: There is some value $c$ in $0..(n+1)$ s.t. $x_i \neq c \ \forall i$
Djikstra's Example

- Assuming a fair execution.
- **Lemma 1**: At least every n rounds $x_0$ changes its value.
- **Lemma 2**: There is some value $c$ in $0..(n+1)$ s.t. $x_i \neq c \ \forall i$
- **Theorem**: Get to all $x_i$s being equal in $O(n^2)$ rounds.
Djikstra's Example

- Assuming a fair execution.
- **Lemma 1**: At least every n rounds $x_0$ changes its value.
- **Lemma 2**: There is some value $c$ in $0..(n+1)$ s.t. $x_i \neq c \ \forall i$
- **Theorem**: Get to all $x_i$s being equal in $O(n^2)$ rounds.
- **Why?**

At process 0

```java
do {
    if (x_4 == x_0) {
        x_0 = (x_0 + 1) % 5;
    }
} while (true);
```

At process n

```java
do {
    if (x_n != x_{n-1}) {
        x_n = x_{n-1};
    }
} while (true);
```
Other Self Stabilizing Algorithms?
Finding a Minimal Spanning Tree
Finding a Minimal Spanning Tree
Finding a Minimal Spanning Tree
Thoughts on how?
Finding a Minimal Spanning Tree

At root
do {
    for n in nbr {
        send(n, <d=0, parent=false>)
    }
} while (true);

At process n
do {
    n[i], parent = recv(nbr i)
    d = min(n)
    parent = find(i s.t. n[i] = d-1)
    send(parent, <d=d, parent=true>)
    for n in nbr {
        if n != parent {
            send(n, <d = d, parent=false>)
        }
    }
} while (true);
Finding a Minimal Spanning Tree

Why does this work?
Finding a Minimal Spanning Tree

Why does this work?
Finding a Minimal Spanning Tree

Why does this work?
Finding a Minimal Spanning Tree

Why does this work?
Why does this work?

Finding a Minimal Spanning Tree
Finding a Minimal Spanning Tree

Why does this work?
Maximal Matching
Maximal Matching
Maximal Matching
How?
partner = null

do {
  broadcast(id, partner)
  collect partners for id
  if partner = null and exists j s.t. partners[j] = id {
    partner = j
  }
  if partner = null and exists j s.t. partners[j] = null {
    partner = j
  }
  if partner = j and partners[j] != id {
    partner = null
  }
} while (true);
Why?
CRDTs
Revisiting RSMs
Revisiting RSMs

Client

Application
Ordering

Client

Application
Ordering

Client

Application
Ordering

Client

Application
Ordering
Revisiting RSMs

Application

Ordering

Application

Ordering

Application

Ordering

Application

Ordering

Client

Client

Client

Client
Revisiting RSMs

Diagram showing the interaction between Application, Ordering, and Client components.
What if we didn't care about ordering

Application

Gossip

Application

Gossip

Application

Gossip

Application

Gossip
What if we didn't care about ordering

![Diagram showing application, gossip, and client components]

- Application
- Gossip
- Client
What if we didn't care about ordering
What if we didn't care about ordering
Challenge: Ensuring correctness despite reordering
Challenge

X

Peer 0

Y

Peer 1

Z

Peer 2

A

Peer 3
Challenge

merge(Z, X)  merge(X, Y)  merge(Y, Z)  merge(X, A)
X            Y            Z            A
Peer 0       Peer 1     Peer 2     Peer 3
Challenge

Peer 0

Peer 1

Peer 2

Peer 3

m(Y, m(Z, X))
merge(Z, X)
X
Peer 0

m(A, m(X, Y))
merge(X, Y)
Y
Peer 1

m(X, m(Y, Z))
merge(Y, Z)
Z
Peer 2

m(Y, m(X, A))
merge(X, A)
A
Peer 3
Challenge

Peer 0  Peer 1  Peer 2  Peer 3

merge(Z, X)  merge(Y, Z)  merge(Y, A)  merge(X, A)
m(Y, m(Z, X))  m(A, m(X, Y))  m(X, m(Y, Z))  m(Y, m(X, A))

merge(Z, X)  merge(X, Y)  merge(Y, Z)  merge(X, A)

merge(X, Y)  m(A, m(X, Y))  m(X, m(Y, Z))  m(Y, m(X, A))
m(A, m(Y, Z))  m(X, m(Y, Z))  m(Y, m(X, A))  m(Y, m(X, A))

merge(Z, X)  m(Y, m(Z, X))  m(A, m(X, Y))  m(A, m(Y, Z))
m(A, m(Y, Z))  m(X, m(Y, Z))  m(Y, m(X, A))  m(Y, m(X, A))

merge(Z, X)  m(Y, m(Z, X))  m(A, m(X, Y))  m(A, m(Y, Z))
m(A, m(Y, Z))  m(X, m(Y, Z))  m(Y, m(X, A))  m(Y, m(X, A))
Challenge

\[
\begin{array}{cccc}
\text{Peer 0} & \text{Peer 1} & \text{Peer 2} & \text{Peer 3} \\
\text{merge}(X, Y) & \text{merge}(Z, X) & \text{merge}(Y, Z) & \text{merge}(X, A) \\
\text{merge}(X, A) & \text{merge}(Y, m(X, A)) & \text{merge}(A, m(X, Y)) & \text{merge}(m(Y, m(X, A))) \\
\text{merge}(A, m(Y, m(Z, X))) & \text{merge}(m(Y, m(Z, X))) & \text{merge}(m(Z, m(A, m(X, Y)))) & \text{merge}(m(Z, m(A, m(X, Y)))) \\
\end{array}
\]

Need all of these to be equal.
Modelling a Merge Function
Modelling a Merge Function

• Treat updates as a set.
Modelling a Merge Function

• Treat updates as a set.

• For previous example \{A, X, Y, Z\}
Modelling a Merge Function

• Treat updates as a set.

• For previous example \{A, X, Y, Z\}

• Define merge function to be the least upper bound (similar to supremum).
Modelling a Merge Function

- Treat updates as a set.
- For previous example \{A, X, Y, Z\}
- Define merge function to be the least upper bound (similar to supremum).
  - Commutative, associative and idempotent.
Modelling a Merge Function

• Treat updates as a set.

• For previous example \{A, X, Y, Z\}

• Define merge function to be the \textbf{least upper bound} (similar to supremum).
  
  • Commutative, associative and idempotent.

• Thus \(\text{LUB}(A, \text{LUB}(X, \text{LUB}(Y, Z)))) = \text{LUB}(X, \text{LUB}(A, \text{LUB}(Y, Z)))) = \ldots\)
Modelling a Merge Function

• Treat updates as a set.
• For previous example \{A, X, Y, Z\}
• Define merge function to be the **least upper bound** (similar to supremum).
  • Commutative, associative and idempotent.
  • Thus \(\text{LUB}(A, \text{LUB}(X, \text{LUB}(Y, Z)))) = \text{LUB}(X, \text{LUB}(A, \text{LUB}(Y, Z)))) = \ldots\)
• In abstract algebra posets with LUBs are called semilattices.
Is this enough?
Modelling a Merge Function
Modelling a Merge Function

• Sufficient for consistency.
Modelling a Merge Function

- Sufficient for consistency.
- Not sufficient to make sure all semantics are preserved.
Modelling a Merge Function

- Sufficient for consistency.
- Not sufficient to make sure all semantics are preserved.
- In particular picking the least upper bound might lose operations.
Counters

$$x = [0, 0, 0]$$  $$x = [0, 0, 0]$$  $$x = [0, 0, 0]$$

Peer 0  Peer 1  Peer 3

Is element wise max a LUB?
Counters

\[ x = [0, 0, 0] \quad x = [0, 0, 0] \quad x = [0, 0, 0] \]

\[ + \]

Peer 0 \quad Peer 1 \quad Peer 3

Is element wise max a LUB?
Counters

\[ x = [1, 0, 0] \quad x = [0, 0, 0] \quad x = [0, 0, 0] \]

+ 
Peer 0  Peer 1  Peer 3

Is element wise max a LUB?
Counters

\[ x = [1, 0, 0] \quad | \quad x = [0, 0, 0] \quad | \quad x = [0, 0, 0] \]

+ Peer 0  Peer 1  - Peer 3

Is element wise max a LUB?
**Counters**

<table>
<thead>
<tr>
<th>x = [1, 0, 0]</th>
<th>x = [0, 0, 0]</th>
<th>x = [0, 0, -1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Peer 0</td>
<td>Peer 1</td>
<td>Peer 3</td>
</tr>
</tbody>
</table>

Is element wise max a LUB?
Counters

$x = [1, 0, 0]$ | $x = [0, 0, 0]$ | $x = [0, 0, -1]$  

+  
Peer 0 | Peer 1 |  

-  
Peer 3

Suppose we use element wise max as LUB.

Is element wise max a LUB?
### Counters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = \text{LUB}([1, 0, 0], [0,0,0], [0,0, -1])$</td>
<td>$x = \text{LUB}([1, 0, 0], [0,0,0], [0,0, -1])$</td>
<td>$x = \text{LUB}([1, 0, 0], [0,0,0], [0,0, -1])$</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>Peer 0</td>
<td>Peer 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Peer 3</td>
</tr>
</tbody>
</table>

Suppose we use element wise max as LUB.

Is element wise max a LUB?
Suppose we use element wise max as LUB.

Is element wise max a LUB?
Modelling a Merge Function

• Need it to be a least upper bound for the state which is a semilattice.
Modelling a Merge Function

- Need it to be a least upper bound for the state which is a semilattice.
- Need LUB to be monotonic.
Modelling a Merge Function

• Need it to be a least upper bound for the state which is a semilattice.

• Need LUB to be monotonic.

• Each application preserves more information.
Build a Collaborative Editor using CRDTs
It was the best of times, it was the poohest of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of credulity, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of credulity, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness
It was the best of times, it was the poohest of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of credulity, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the very worst of belief, it was the season of Darkness.
it was the poorest
It was the poorest.
it was the poorest
it was the poorest
It was the poorest.
it was the poorest
it was the poorest
It was the poorest.
It was the poorest.
It was the poorest way.
It was the poorest.
It was the poorest.
It was the poorest very worse.
It was the poorest very worst.
It was the poorest very worst.
Sometimes it fails.
What consistency guarantees do both of today's mechanisms provide?
When are these good?