Distributed Systems: Class 1

Aurojit Panda
Please interrupt
Please interrupt

When things seem unclear or we are going too fast.
Administrivia
Initial Bits

• If enrolled, you should have Piazza access. Come up if you do not.

• Waitlist: There is still a waitlist.
  
  • If you are going to drop the class do it early.
  
  • Preference will be given to CS PhD students first.

• Webpage is up, posted link on Piazza. Ask around if you don't know where.

• All policies that follow are documented on the webpage.
Personel

- There is no TA at the moment.
Personel

• There is no TA at the moment.

• My office hours are on Thursday from 5-6 PM.
Personel

• There is no TA at the moment.

• My office hours are on Thursday from 5-6 PM.

• Can also meet by appointment.
Personel

• There is no TA at the moment.
• My office hours are on Thursday from 5-6 PM.
  • Can also meet by appointment.
• Send all project questions, etc. to Piazza.
Personel

• There is no TA at the moment.

• My office hours are on Thursday from 5-6 PM.
  • Can also meet by appointment.

• Send all project questions, etc. to Piazza.
  • Likely to get other people to help you.
Overview

- This is a graduate course in distributed systems.
This is a graduate course in distributed systems.

Not assuming much prior knowledge.
Overview

• This is a graduate course in distributed systems.
  • Not assuming much prior knowledge.
  • Assuming prior programming experience.
Overview

• This is a graduate course in distributed systems.
  • Not assuming much prior knowledge.
  • Assuming prior programming experience.
  • Assuming that you can quickly pick up a new programming language.
Overview (continued)

• This version of the class is a bit different -- my first time teaching it.
Overview (continued)

• This version of the class is a bit different -- my first time teaching it.

• A larger focus on classical papers in distributed systems
Overview (continued)

• This version of the class is a bit different -- my first time teaching it.
  
  • A larger focus on classical papers in distributed systems
  
  • Class involves a fair amount of reading -- see papers on website.
Overview (continued)

- This version of the class is a bit different -- my first time teaching it.
- A larger focus on classical papers in distributed systems
- Class involves a fair amount of reading -- see papers on website.
- Skimming through proofs, etc. is fine for the reading.
Overview (continued)

• This version of the class is a bit different -- my first time teaching it.

• A larger focus on classical papers in distributed systems

• Class involves a fair amount of reading -- see papers on website.

• Skimming through proofs, etc. is fine for the reading.

• Please do the reading though, allows you to contribute to class.
Overview (continued)

• This version of the class is a bit different -- my first time teaching it.
  
  • A larger focus on classical papers in distributed systems
  
  • Class involves a fair amount of reading -- see papers on website.
    
    • Skimming through proofs, etc. is fine for the reading.
    
    • Please do the reading though, allows you to contribute to class.

• Require paper summaries to encourage reading.
Grading

• Paper Summaries: 20%
• Class Participation: 7.5%
• In class quizzes (4): 7.5%
• Labs: 25%
• Final Project: 40%
Paper Summaries

- Due at noon every Wednesday (before class).
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  • What problem is being addressed?
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  • What problem is being addressed?
  • What assumptions do they make when addressing this problem?
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  
  • What problem is being addressed?
  
  • What assumptions do they make when addressing this problem?
  
  • What is the proposed solution (at a high level)?
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  • What problem is being addressed?
  • What assumptions do they make when addressing this problem?
  • What is the proposed solution (at a high level)?
  • How would you extend the paper?
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  • What problem is being addressed?
  • What assumptions do they make when addressing this problem?
  • What is the proposed solution (at a high level)?
  • How would you extend the paper?
  • Anything that was hard to understand?
Paper Summaries

• Due at noon every Wednesday (before class).

• Must cover the following points for each paper:
  • What problem is being addressed?
  • What assumptions do they make when addressing this problem?
  • What is the proposed solution (at a high level)?
  • How would you extend the paper?
  • Anything that was hard to understand?

• Try and keep the summaries succinct.
Paper Summaries

Skip up to 6 papers without penalty
Class Participation

• Speak up in class: ask questions, answer questions, express confusion.
Class Participation

• Speak up in class: ask questions, answer questions, express confusion.

• Ideally class should be entirely discussions: keeps it more entertaining.
Class Participation

- Speak up in class: ask questions, answer questions, express confusion.
- Ideally class should be entirely discussions: keeps it more entertaining.
- Please introduce yourself when speaking.
Class Participation

• Speak up in class: ask questions, answer questions, express confusion.

• Ideally class should be entirely discussions: keeps it more entertaining.

• Please introduce yourself when speaking.

• Answer questions on Piazza.
Class Participation

• Speak up in class: ask questions, answer questions, express confusion.

• Ideally class should be entirely discussions: keeps it more entertaining.

• Please introduce yourself when speaking.

• Answer questions on Piazza.

• Come to office hour and ask for participation credit.
Class Participation

• Speak up in class: ask questions, answer questions, express confusion.

• Ideally class should be entirely discussions: keeps it more entertaining.

• Please introduce yourself when speaking.

• Answer questions on Piazza.

• Come to office hour and ask for participation credit.

• Reserve the right to ask you why not participate.
In Class Quiz

• We will have four quizzes.
In Class Quiz

- We will have four quizzes.
- Held in the last 20 minutes of class.
In Class Quiz

• We will have four quizzes.
• Held in the last 20 minutes of class.
• Both multiple choice and longform answer.
In Class Quiz

• We will have four quizzes.
• Held in the last 20 minutes of class.
• Both multiple choice and longform answer.
• Must e-mail me if you will miss a quiz.
In Class Quiz

• We will have four quizzes.
• Held in the last 20 minutes of class.
• Both multiple choice and longform answer.
• Must e-mail me if you will miss a quiz.
  • They are marked in the class schedule.
Labs

- Traditional programming assignments in Go, using GRPC, etc.
Labs

• Traditional programming assignments in Go, using GRPC, etc.

• The first one is out now.
Labs

- Traditional programming assignments in Go, using GRPC, etc.
- The first one is out now.
  - Mostly focused on getting you started.
Labs

- Traditional programming assignments in Go, using GRPC, etc.
- The first one is out now.
  - Mostly focused on getting you started.
  - Currently due in a week.
Labs

• Traditional programming assignments in Go, using GRPC, etc.

• The first one is out now.
  • Mostly focused on getting you started.
  • Currently due in a week.

• We will have between 2--3 labs.
Labs

• Traditional programming assignments in Go, using GRPC, etc.

• The first one is out now.
  • Mostly focused on getting you started.
  • Currently due in a week.

• We will have between 2--3 labs.

• Note: I might move dates when labs are out and due around. Sorry!
Labs

- Traditional programming assignments in Go, using GRPC, etc.
- The first one is out now.
  - Mostly focused on getting you started.
  - Currently due in a week.
- We will have between 2--3 labs.
- Note: I might move dates when labs are out and due around. Sorry!
- **Late Policy**: Up to 3 days late across all labs.
Final Project

• You need to work on a substantial project in this class.
Final Project

• You need to work on a substantial project in this class.

• If you are already involved in a research project...
Final Project

- You need to work on a substantial project in this class.
- If you are already involved in a research project...
  - ... it is fine to figure out a way to use it in this class.
Final Project

• You need to work on a substantial project in this class.

• If you are already involved in a research project...
  • ... it is fine to figure out a way to use it in this class.

• If not, that is fine too.
Final Project

• You need to work on a substantial project in this class.

• If you are already involved in a research project...
  • ... it is fine to figure out a way to use it in this class.

• If not, that is fine too.

• Not looking for a research project alone.
Final Project

• You need to work on a substantial project in this class.

• If you are already involved in a research project...
  • ... it is fine to figure out a way to use it in this class.

• If not, that is fine too.

• Not looking for a research project alone.
  • Can also implement an existing paper.
Final Project

• You need to work on a substantial project in this class.
• If you are already involved in a research project...
  • ... it is fine to figure out a way to use it in this class.
• If not, that is fine too.
• Not looking for a research project alone.
  • Can also implement an existing paper.
  • Extend an existing project (e.g., an open source project).
Final Project

• You need to work on a substantial project in this class.

• If you are already involved in a research project...
  • ... it is fine to figure out a way to use it in this class.

• If not, that is fine too.

• Not looking for a research project alone.
  • Can also implement an existing paper.
  • Extend an existing project (e.g., an open source project).
  • Extend one of the previous labs.
Final Project

• The project should not be an impediment to taking this class.
Final Project

• The project should not be an impediment to taking this class.

• Aim is to provide a way for students to get their hands dirty.
Final Project

• The project should not be an impediment to taking this class.

• Aim is to provide a way for students to get their hands dirty.

• If the project worries you come talk to me we will find something for you to do.
Providing Feedback

• This is my first time doing teaching a class without training wheels.
Providing Feedback

- This is my first time doing teaching a class without training wheels.
- Unlikely that no one will be unhappy throughout the semester.
Providing Feedback

• This is my first time doing teaching a class without training wheels.

• Unlikely that no one will be unhappy throughout the semester.

• If you are unhappy you can try and fix things:
Providing Feedback

- This is my first time doing teaching a class without training wheels.
- Unlikely that no one will be unhappy throughout the semester.
- If you are unhappy you can try and fix things:
  - Complain in office hours, or over e-mail.
Providing Feedback

• This is my first time doing teaching a class without training wheels.

• Unlikely that no one will be unhappy throughout the semester.

• If you are unhappy you can try and fix things:
  • Complain in office hours, or over e-mail.
  • Google form linked in course policy for anonymous complaints.
Providing Feedback

• This is my first time doing teaching a class without training wheels.

• Unlikely that no one will be unhappy throughout the semester.

• If you are unhappy you can try and fix things:
  • Complain in office hours, or over e-mail.
  • Google form linked in course policy for anonymous complaints.

• Promise to listen to all complaints, not necessarily act on them.
One last bit of Administrivia...
Academic Honesty

• Sadly this needs to be said.
Academic Honesty

- Sadly this needs to be said.

- Please don't plagiarize (projects, quizzes, whatever), and please cite your sources
Academic Honesty

• Sadly this needs to be said.
• Please don't plagiarize (projects, quizzes, whatever), and please cite your sources
• First time teaching, first semester at NYU.
Academic Honesty

• Sadly this needs to be said.

• Please don't plagiarize (projects, quizzes, whatever), and please cite your sources

• First time teaching, first semester at NYU.

• Really think highly of everyone here, don't shatter my illusion.
Academic Honesty

• Sadly this needs to be said.

• Please don't plagiarize (projects, quizzes, whatever), and please cite your sources

• First time teaching, first semester at NYU.

  • Really think highly of everyone here, don't shatter my illusion.

• If you need help, need better grades for some reason ... come see me.
Academic Honesty

• Sadly this needs to be said.

• Please don't plagiarize (projects, quizzes, whatever), and please cite your sources.

• First time teaching, first semester at NYU.

  • Really think highly of everyone here, don't shatter my illusion.

• If you need help, need better grades for some reason ... come see me.

  • We can figure out a way to fix this sort of thing.
Academic Honesty

• Sadly this needs to be said.

• Please don't plagiarize (projects, quizzes, whatever), and please cite your sources.

• First time teaching, first semester at NYU.

• Really think highly of everyone here, don't shatter my illusion.

• If you need help, need better grades for some reason ... come see me.

• We can figure out a way to fix this sort of thing.

• If you get caught, I will use every instrument at my disposal.
Distributed Systems
What is a distributed system?
Main Characteristics (for us)

- Comprised of several processes, each running on its own "computer"
  - For the purposes of this class node.
- Time for processes to communicate is significant compared to an instruction.
- **Partial failures**: some processes can fail while others remain alive.
  - In contrast to traditional programs where everything fails or nothing does.
Assumptions: Message Passing

• Only considering *message* passing.

• Processes communicate by sending each other messages.

  Process 0  Process 1

• Other models (shared memory) can be shown to be equivalent.
Assumptions: Message Passing

- Only considering *message* passing.

- Processes communicate by sending each other messages.

  ![Diagram showing Ping between Process 0 and Process 1]

- Other models (shared memory) can be shown to be equivalent.
Assumptions: Message Passing

- Only considering **message** passing.

- Processes communicate by sending each other messages.

  ![Diagram](Diagram)

- Other models (shared memory) can be shown to be equivalent.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages from different processes can be reordered.
Assumptions: Async Network

- **Asynchronous Network**: Messages can be arbitrarily delayed.
Assumptions: Async Network

- **Asynchronous Network**: Messages can be lost.
Assumptions: Fail Stop

- **Fail Stop**: Failed processes do not send messages.
Assumptions: Fail Stop

- **Fail Stop:** Failed processes do not send messages.
Assumptions: Fail Stop

- **Fail Stop**: Failed processes do not send messages.
Why distributed system?
Three Main Reasons
Three Main Reasons

• Fault tolerance
Three Main Reasons

• Fault tolerance

• Survive some forms of failures or bug.
Three Main Reasons

• Fault tolerance
  • Survive some forms of failures or bug.

• Scalability
Three Main Reasons

- Fault tolerance
  - Survive some forms of failures or bug.
- Scalability
  - Use more resources than a single computer can provide.
Three Main Reasons

• Fault tolerance
  • Survive some forms of failures or bug.

• Scalability
  • Use more resources than a single computer can provide.

• Geographic Reach
Three Main Reasons

• Fault tolerance
  • Survive some forms of failures or bug.

• Scalability
  • Use more resources than a single computer can provide.

• Geographic Reach
  • Work even when information is spread across large distances.
How to reason about distributed systems?
Reasoning about Systems

- Correctness: Specify some invariants that are held (explained next).
- Performance: How fast does it do whatever it is it does?
  - Systems view: Look at empirical measurements under various circumstances.
  - Theoretical view: Look at asymptotic number of messages.
Safety

- **Safety**: Something bad does not happen.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.

This class has no exams.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

• **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invariant: I will never contradict a previously announced class policy.
Safety

- **Safety**: Something bad does not happen.

Invarient: I will never contradict a previously announced class policy.
Liveness

- **Liveness**: Eventually something good will happen.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

• **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

• **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

• **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
• **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
• **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

• **Liveness**: Eventually something good will happen.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Liveness

- **Liveness**: Eventually something good will happen.

Invariant: I will eventually return any pen that you lend me.
Why are distributed systems hard?
Several Problems

• Reasoning about what happens during failures.
• Deciding whether a node has actually failed.
• Agreeing on an order of events.
Several Problems

• Reasoning about what happens during failures.
• Deciding whether a node has actually failed.
• Agreeing on an order of events.
Why Event Ordering Matters

• Natural way to think about programs and instructions.

• For example:

  Process X
  Q.send(add X)
  Y.send(I am in queue)

  Process Y
  msg = wait()
  ...
  Q.send(add Y)

  Process Q
  while(true)
  msg = wait()
  queue.add(msg)
Why Event Ordering Matters

• Natural way to think about programs and instructions.

• For example:

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
  msg = wait()
  queue.add(msg)

What are the possible values of queue at the end of this?
Why Event Ordering Matters

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
  msg = wait()
  queue.add(msg)

What are the possible values of queue at the end of this?

X
Y

X

Y

Y

X
Why Event Ordering Matters

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
   msg = wait()
   queue.add(msg)

What are the possible values of queue at the end of this?

X Y
X Y
X Y

X Y
X Y
X Y

X Y
X Y
X Y
Why Event Ordering Matters

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
msg = wait()
queue.add(msg)

What are the possible values of queue at the end of this?

X Y X
Y X Y
Why Event Ordering Matters

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
msg = wait()
queue.add(msg)

What are the possible values of queue at the end of this?

Causally Compatible
Why Event Ordering Matters

Process X
Q.send(add X)
Y.send(I am in queue)

Process Y
msg = wait()
...
Q.send(add Y)

Process Q
while(true)
msg = wait()
queue.add(msg)

What are the possible values of queue at the end of this?

Causally Compatible

Not Causally Compatible
How to Order Events?

![Graph showing Q, X, and Y ordered in time]

- Time: 1, 3, 8, 9, 20, 22
- Events: Q, X, Y
How to Order Events?

Event | Time
---|---
send(•) | 1
send(•) | 3
receive(○) | 8
send(•) | 9
receive(○) | 20
receive(○) | 22
How to Order Events?

- Real time provides a way to order events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>1</td>
</tr>
<tr>
<td>send()</td>
<td>3</td>
</tr>
<tr>
<td>receive()</td>
<td>8</td>
</tr>
<tr>
<td>send()</td>
<td>9</td>
</tr>
<tr>
<td>receive()</td>
<td>20</td>
</tr>
<tr>
<td>receive()</td>
<td>22</td>
</tr>
</tbody>
</table>
How to Order Events?

- Real time provides a way to order events.

- **Total order**: All events are ordered wrt each other.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>1</td>
</tr>
<tr>
<td>send()</td>
<td>3</td>
</tr>
<tr>
<td>receive()</td>
<td>8</td>
</tr>
<tr>
<td>send()</td>
<td>9</td>
</tr>
<tr>
<td>receive()</td>
<td>20</td>
</tr>
<tr>
<td>receive()</td>
<td>22</td>
</tr>
</tbody>
</table>
How to Order Events?

- Real time provides a way to order events.
  - **Total order**: All events are ordered wrt each other.
  - Order is compatible with causality.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>1</td>
</tr>
<tr>
<td>send()</td>
<td>3</td>
</tr>
<tr>
<td>receive()</td>
<td>8</td>
</tr>
<tr>
<td>send()</td>
<td>9</td>
</tr>
<tr>
<td>receive()</td>
<td>20</td>
</tr>
<tr>
<td>receive()</td>
<td>22</td>
</tr>
</tbody>
</table>
How to Order Events?

- Real time provides a way to order events.
- **Total order**: All events are ordered wrt each other.
- Order is compatible with causality.
- Can use this ordering to ensure queue is causally compatible.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>1</td>
</tr>
<tr>
<td>send()</td>
<td>3</td>
</tr>
<tr>
<td>receive()</td>
<td>8</td>
</tr>
<tr>
<td>send()</td>
<td>9</td>
</tr>
<tr>
<td>receive()</td>
<td>20</td>
</tr>
<tr>
<td>receive()</td>
<td>22</td>
</tr>
</tbody>
</table>
How to Order Events?

• Real time provides a way to order events.
  • **Total order**: All events are ordered wrt each other.
  • Order is compatible with causality.
  • Can use this ordering to ensure queue is causally compatible.
  • How?

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>1</td>
</tr>
<tr>
<td>send()</td>
<td>3</td>
</tr>
<tr>
<td>receive()</td>
<td>8</td>
</tr>
<tr>
<td>send()</td>
<td>9</td>
</tr>
<tr>
<td>receive()</td>
<td>20</td>
</tr>
<tr>
<td>receive()</td>
<td>22</td>
</tr>
</tbody>
</table>
How to Order Events?

• Time sounds simple to use, but is hard to implement.
How to Order Events?

- Time sounds simple to use, but is hard to implement.

- To use time as above, all *processes* need to agree on time.
How to Order Events?

• Time sounds simple to use, but is hard to implement.

• To use time as above, all *processes* need to agree on time.

• Must make sure *events* happen at a frequency lower than clock ticks.
How to Order Events?

- Time sounds simple to use, but is hard to implement.
- To use time as above, all **processes** need to agree on time.
- Must make sure **events** happen at a frequency lower than clock ticks.
- Make sure **clock** does not diverge over time.
How to Order Events?

• Time sounds simple to use, but is hard to implement.

• To use time as above, all processes need to agree on time.

• Must make sure events happen at a frequency lower than clock ticks.

• Make sure clock does not diverge over time.

• None of this happens in real life.
How to Order Events?

- Time sounds simple to use, but is hard to implement.
- To use time as above, all **processes** need to agree on time.
- Must make sure **events** happen at a frequency lower than clock ticks.
- Make sure **clock** does not diverge over time.
- None of this happens in real life.
  - Problems with both physics and engineering.
What Do We Want?

• Preserve causal order: related events should appear to happen in order.
What Do We Want?

• Preserve causal order: related events should appear to happen in order.

• Not sure about other events: they can happen in arbitrary order.
What Do We Want?

- Preserve causal order: related events should appear to happen in order.
- Not sure about other events: they can happen in arbitrary order.
- Do not rely on a synchronized clock or a source of time.
Why Causal Order?

• As an aside: why causal order?

• Idea: Causality indicates when one event could have affected another.

\[ \text{Q} \rightarrow \text{X} \rightarrow \text{Y} \]
Why Causal Order?

- As an aside: why causal order?

- Idea: Causality indicates when one event could have affected another.
Why Causal Order?

• As an aside: why causal order?

• Idea: Causality indicates when one event could have affected another.

![Diagram showing causality between Q, X, and Y]
Why Causal Order?

• As an aside: why causal order?

• Idea: Causality indicates when one event could have affected another.
When Are Events Causally Linked
When Are Events Causally Linked

- Events in the same process
When Are Events Causally Linked

- Events in the same process
- A message send and receive
When Are Events Causally Linked

- Events in the same process
- A message send and receive
- Transitive closure of the previous.

Why?
Happens Before

• A way to order causally linked events.

• **Partial order**: cannot determine ordering for events that are not causally linked.

• Assumes that we can determine an order of events for a process
  • Simple: just increment a counter whenever an event occurs.
Happens Before
Happens Before

\[ e_1 \rightarrow e_2 \quad e_2 \rightarrow e_3 \quad e_1 \rightarrow e_3 \]
Happens Before

\[ e_1 \rightarrow e_2 \quad e_2 \rightarrow e_3 \quad e_1 \rightarrow e_3 \]

\[ e_1 \rightarrow e_2 \]

\[ e_1 \rightarrow e_3 \]
Happens Before

$e_1 \rightarrow e_2 \quad e_2 \rightarrow e_3 \quad e_1 \rightarrow e_3$

$e_1 \rightarrow e_2$

$e_1 \rightarrow e_2 \quad e_2 \rightarrow e_3 \quad e_1 \rightarrow e_3$
Approximating Happens Before

- Can we approximate happens before, i.e.:
Approximating Happens Before

• Can we approximate happens before, i.e.:

  • Define a set of functions \( c_i \) one for each process \( i \).
Approximating Happens Before

• Can we approximate happens before, i.e.:

  • Define a set of functions $c_i$ one for each process $i$.
  
  • And a single function $c$ such that
Approximating Happens Before

• Can we approximate happens before, i.e.:

  • Define a set of functions $c_i$ one for each process $i$.
  
  • And a single function $c$ such that

$$c(e) = c_i(e) \quad \text{If event } e \text{ occurred at process } i$$
Approximating Happens Before

- Can we approximate happens before, i.e.:
  - Define a set of functions $c_i$ one for each process $i$.
  - And a single function $c$ such that
    
    \[
    c(e) = c_i(e) \quad \text{If event } e \text{ occurred at process } i \\
    e_0 \rightarrow e_1 \implies c(e_0) < c(e_1)
    \]
Approximating Happens Before

• Can we approximate happens before, i.e.:

  • Define a set of functions $c_i$ one for each process $i$.

  • And a single function $c$ such that

  \[ c(e) = c_i(e) \quad \text{If event } e \text{ occurred at process } i \]

  \[ e_0 \rightarrow e_1 \implies c(e_0) < c(e_1) \]

  Lamport says yes!
Lamport Clock

- Each process $i$ keeps a counter $c_i$.

\[ c_j = \max(v, c_j + 1) \]
Lamport Clock

- Each process $i$ keeps a counter $c_i$.
- When sending a message process $i$ increments $c_i$.

\[ c_j = \max(v, c_j + 1) \]
Lamport Clock

• Each process $i$ keeps a counter $c_i$.

• When sending a message process $i$ increments $c_i$.

  • Also includes new value of $c_i$ in the message.

\[ c_j = \max(v, c_j + 1) \]
Lamport Clock

• Each process $i$ keeps a counter $c_i$.

• When sending a message process $i$ increments $c_i$.
  
  • Also includes new value of $c_i$ in the message.

• When some process $j$ receives a message $m$ with clock value $v$ it:

  $$c_j = \max(v, c_j + 1)$$
Lamport Clock

• Each process $i$ keeps a counter $c_i$.

• When sending a message process $i$ increments $c_i$.
  • Also includes new value of $c_i$ in the message.

• When some process $j$ receives a message $m$ with clock value $v$ it:

  \[ c_j = \max(v, c_j + 1) \]
Lamport Clock

- Each process $i$ keeps a counter $c_i$.
- When sending a message process $i$ increments $c_i$.
- Also includes new value of $c_i$ in the message.
- When some process $j$ receives a message $m$ with clock value $v$ it:
  \[ c_j = \max(v, c_j + 1) \]
- For all other operations (processing, etc.) process $i$ increments $c_i$. 
Lamport Clock (contd)

• Then for any event $e$, $c_i(e) = c_i$ immediately after process $i$ has handled $e$. 
Lamport Clock (contd)

- Then for any event \( e \), \( c_i(e) = c_i \) immediately after process \( i \) has handled \( e \).

- Does this work?
Lamport Clock (contd)

• Then for any event $e$, $c_i(e) = c_i$ immediately after process $i$ has handled $e$.

• Does this work?

• What can you say about two events $e_0$ and $e_1$ that are not causally linked?
  
  $e_0 \not\rightarrow e_1$
Can we do better?
Summary

• One source of our problems is the use of asynchronous networks.
  • Makes it hard to maintain ordering across events.
  • Lack of ordering makes it hard to reason about programs.
• We can use a variety of mechanisms to recover causal ordering.
  • Unclear if there are ways to recover actual (real time) ordering.
• Almost all of the semester is about looking at this problem in different settings.
Reminders

• Please look through Lab 0

• Post on Piazza, etc. if you run into trouble.

• Read papers for next week, send me summaries.

• E-mail me if you have any concerns about grading, the project, etc.