DISTRIBUTED CONSENSUS-
PART 2: REPLICATION ABSTRACTION
AND ASYNCHRONOUS LEARNERS

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
MODELLING OF PROBLEM THROUGH ABSTRACTION

Coupled Process

Object $O^1$: Value$^1$

Object $O^2$: Value$^2$

Coupled Process

Object $O^1$: Value$^1$

Object $O^2$: Value$^2$
Modeling of Problem through Abstraction

Coupled Process

O¹ = Value¹²

p.send(Value¹²)

Coupled Process

O¹ = Value¹¹

p.receive(O¹)

Local Assignment

Remote Assignment

Replicated Assignment?

Coupled processed may not be full replicas

State shared may be a command or procedure call

Replicated RPC
**Consensus Abstraction: Initial Operations**

Consensus Mechanism `<StateType>`

- `StateType getConsensusState();`
- `StateType getLastConsensusState();`
- `float propose(StateType aProposal);`
- `ProposalState waitForConsensus (float aProposalNumber);`
- `void addConsensusListener(ConsensusListener<StateType> aConsensusListener);`
- `boolean isPending(float aProposalNumber);`
- `boolean someProposalIsPending();`
**Consensus Listener**

- `void newConsensusState(StateType aState);`
- `void newProposalState(float aProposalNumber, StateType aProposal, ProposalState aProposalState);`
- `void newLocalProposalState(float aProposalNumber, StateType aProposal, ProposalState aProposalState);`
- `void newRemoteProposalState(float aProposalNumber, StateType aProposal, ProposalState aProposalState);`
**Consensus Abstraction: Consensus State**

```java
public StateType getLastConsensusState() {
    return consensusState;
}

public StateType getConsensusState() {
    if (someProposalIsPending())
        return null;
    return consensusState;
}

protected synchronized void newProposalState(float aProposalNumber,
                                           StateType aState,
                                           ProposalState aProposalState) {
    if (!isPending(aProposalNumber)) {
        return;
    }
    proposalState.put(aProposalNumber, aProposalState);
    notify(aProposalNumber, aState, aProposalState);
}
```
**Consensus Abstraction: Wait/Notify**

```java
protected synchronized void notify(float aProposalNumber, StateType aState, ProposalState aProposalState) {
    notifyListeners(aProposalNumber, aState, aProposalState);
    notifyAll();
}
```

```java
public synchronized ProposalState waitForConsensus(float aProposalNumber) {
    while (isPending(aProposalNumber)) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    ProposalState returnValue = proposalState.get(aProposalNumber);
    return returnValue;
}
```
SIMPLEST USEFUL ALGORITHM?

- Multiple client UIs commit to single server
  - Browser-Sakai
- Nested transaction involving multiple logical servers
  - Travelocity
- Physical replication with multiple changers
  - Diff-based with divergence (Git)
  - Snapshot-based (Google Drive, OneDrive)
  - Command-based: replicated state machines (Google Docs, LiveMeeting)
- Lock and other meta/configuration state
  - Live Meeting
- Physical mirroring
  - Akamai
- Master (primary)-slave (backup) replication
- Master-master replication
  - Disjoint writes
  - Overlapping writes
ASYNCHRONOUS ALGORITHM

```java
protected void propose(float aProposalNumber, StateType aProposal) {
    ProposalRejectionKind aRejectionKind = ProposalRejectionKind.ACCEPTED;
    recordSentLearnNotification(aProposalNumber, aProposal, aRejectionKind);
    sendLearnNotification(aProposalNumber, aProposal, aRejectionKind);
}

public synchronized void learn(float aProposalNumber, StateType aProposal, ProposalRejectionKind aRejectionKind) {
    recordReceivedLearnNotification (aProposalNumber, aProposal, aRejectionKind);
}
```
# Consensus Abstraction: Initial Operations (Review)

<table>
<thead>
<tr>
<th>Consensus Mechanism  &lt;StateType&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateType getConsensusState();</td>
</tr>
<tr>
<td>StateType getLastConsensusState();</td>
</tr>
<tr>
<td>float propose(StateType aProposal);</td>
</tr>
<tr>
<td>ProposalState waitForConsensus (float aProposalNumber);</td>
</tr>
<tr>
<td>void addConsensusListener(ConsensusListener&lt;StateType&gt; aConsensusListener);</td>
</tr>
<tr>
<td>boolean isPending(float aProposalNumber);</td>
</tr>
<tr>
<td>boolean someProposalIsPending();</td>
</tr>
</tbody>
</table>
SIMPLEST USEFUL ALGORITHM?

- Multiple client UIs commit to single server
  - Browser-Sakai
- Nested transaction involving multiple logical servers
  - Travelocity
- Physical replication with multiple changers
  - Diff-based with divergence (Git)
  - Snapshot-based (Google Drive, OneDrive)
  - Command-based: replicated state machines (Google Docs, LiveMeeting)
- Lock and other meta/configuration state
  - Live Meeting
- Physical mirroring
  - Akamai
- Master (primary)-slave(backup) replication
- Master-master replication
  - Disjoint unrelated writes to different sequencing masters
  - Overlapping writes
Algorithm-Independent Propose (Value)

1. Get proposal number
2. Propose (proposal number, value)
3. Record state and return proposal number

- Get proposal number and propose with proposal number are changeable
- Top-level method is reusable
PROPOSE (PROPOSAL NUMBER, VALUE)

Send proposal with proposal number and value to all learners (usually includes self)

E.g. learners (mirrors, slaves)
LEARN (PROPOSAL NUMBER, VALUE, VERDICT)

Change consensus value
**Asynchronous Algorithm (Proposer)**

```java
public float propose(StateType aProposal) {
    float myProposalNumber = getAndSetNextProposalNumber(aProposal);
    recordProposal(myProposalNumber, aProposal);
    dispatchPropose(float(myProposalNumber, aProposal);
    return myProposalNumber;
}

protected void dispatchPropose(float aProposalNumber, StateType aProposal) {
    localPropose(aProposalNumber, aProposal);
}

protected void localPropose(float aProposalNumber, StateType aProposal) {
    ProposalFeedbackKind aFeedbackKind = ProposalFeedbackKind.SUCCESS;
    recordAndSendLearnNotification(aProposalNumber, aProposal, aFeedbackKind);
}
```
Asynchronous Algorithm (Learner)

```java
public synchronized void learn(float aProposalNumber, StateType aProposal, ProposalFeedbackKind aFeedbackKind) {
    recordReceivedLearnNotification
    (aProposalNumber, aProposal, aFeedbackKind);
}

protected void recordReceivedLearnNotification(float aProposalNumber,
                                            StateType aProposal, ProposalFeedbackKind aFeedbackKind) {
    recordProposalState(aProposalNumber, aProposal);
    if (!isPending(aProposalNumber))
        return;
    maxProposalNumberReceivedInLearnNotification = Math.max(
        maxProposalNumberReceivedInLearnNotification, aProposalNumber);
    newProposalState(aProposalNumber, aProposal,
                    toProposalState(aProposalNumber, aProposal, aFeedbackKind));
}
```

Feedback kind for future algorithms

Proposal number not used until we reach Paxos
EXAMPLE SCENARIO

1

proposeMeaning(MEANING_1);

2

proposeMeaning(MEANING_2);
Abstraction Use (Learner and Proposers)

```java
protected ConsensusMechanism<Integer>
    createLocalMeaningOfLifeMechanism(short anId) {
        return meaningConsensusMechanismFactory().createConsensusMechanism(
            SESSION_MANAGER_HOST,
            EXAMPLE_SESSION,
            memberId, portNumber, MEANING_OF_LIFE_CONSENSUS_MECHANISM_NAME,
            sessionChoice, numMembersToWaitFor());
    }
```

```java
public class AMeaningOfLifeConsensusListener implements ConsensusListener<Integer> {

    public void newConsensusState(Integer aState) {
        System.out.println("Meaning of Life:" + aState);
    }

    ...
}
```
Abstraction Use (Proposer)

```java
public void proposeMeaning(Integer aValue) {
    while (true) {
        if (meaningOfLifeMechanism.someProposalIsPending()) {
            meaningOfLifeMechanism.waitForConsensus(
                meaningOfLifeMechanism.lastProposalNumber());
        }

        float aMeaningOfLifeProposal =
            meaningOfLifeMechanism.propose(aValue);
        meaningOfLifeMechanism.waitForConsensus(aMeaningOfLifeProposal);
        ProposalState aState = meaningOfLifeMechanism
            .getProposalState(aMeaningOfLifeProposal);
        if (aState != ProposalState.PROPOSAL_CONCURRENT_OPERATION) {
            break;
        }
    }
}
```
**Example Scenario**

1. `proposeMeaning(MEANING_1);`

2. `proposeMeaning(MEANING_2);`
ASYNCHRONOUS ALGORITHM

<table>
<thead>
<tr>
<th>Meaning of Life</th>
<th>Value</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>Consensus</td>
</tr>
</tbody>
</table>

No Atomic Broadcast
Actual Run (Proposers and Learners)

1

NonAtomicNonVetoableProposer1Launcher [Java Application] D:\Program Files\Java\jdk1
I***Tracer: showInfo = true
I*** (ProposalLearnNotificationSent) Meaning, 1.0001 = 42 --> NC
I*** (ProposalLearnNotificationReceived) Meaning, 1.0002 = 29 -
I*** (ProposalStateChanged) Meaning, 1.0002 = 29 --> PROPOSAL_CC
Meaning of Life: 29
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001 = 42 -
I*** (ProposalStateChanged) Meaning, 1.0001 = 42 --> PROPOSAL_CC
Meaning of Life: 42

2

NonAtomicNonVetoableProposer2Launcher [Java Application] D:\Program Files\Java\jdk1
I***Tracer: showInfo = true
I*** (ProposalLearnNotificationSent) Meaning, 1.0002 = 29 --> NC
I*** (ProposalLearnNotificationReceived) Meaning, 1.0002 = 29 -
I*** (ProposalStateChanged) Meaning, 1.0002 = 29 --> PROPOSAL_CC
Meaning of Life: 29
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001 = 42 -
I*** (ProposalStateChanged) Meaning, 1.0001 = 42 --> PROPOSAL_CC
Meaning of Life: 42
Actual Run (Learners Only)

NonAtomicLearner3Launcher [Java Application] D:\Program Files\Java\jdk1.8.0_40\bin\java I***Tracer: showInfo = true I*** (ProposalLearnNotificationReceived) Meaning, 1.0002=29--->PROPOSAL_CON Meaning of Life: 29 I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42--->PROPOSAL_CON Meaning of Life: 42

NonAtomicLearner3Launcher [Java Application] D:\Program Files\Java\jdk1.8.0_40\bin\java I***Tracer: showInfo = true I*** (ProposalLearnNotificationReceived) Meaning, 1.0002=29--->PROPOSAL_CON Meaning of Life: 29 I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42--->PROPOSAL_CON Meaning of Life: 42
ASYNCHRONOUS CONSISTENCY SCENARIOS

- Multiple client UIs commit to single server
  - Browser-Sakai

- Nested transaction involving multiple logical servers
  - Travelocity (non replicated)

- Physical replication with multiple changers
  - Diff-based with divergence (Git)
  - Snapshot-based (Google Drive, OneDrive)
  - Command-based: replicated state machines (Google Docs, LiveMeeting)

- Lock and other meta/configuration state
  - Live Meeting

- Physical mirroring
  - Akamai

- Master (primary)-slave (backup) replication

- Master-master replication
  - Disjoint writes
  - Overlapping writes