DISTRIBUTED CONSENSUS-
PART 3: (SEMI) SYNCHRONOUS
REPLICATION AND ACCEPTORS

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
ASYNCHRONOUS CONSISTENCY SCENARIOS (LAST SLIDE OF PART 2)

- 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
LEADING-FOLLOWING GENERAL PROBLEM: IMMEDIATE CONSISTENCY

If reliable communication, know messenger will eventually reach destination (eventual consistency, asynchronous replication)

Do not know when messenger reaches (problem compounded with n receivers)

Do not know if general (process) has looked at message
**ASYNCHRONOUS ALGORITHM**

```java
protected void localPropose(float aProposalNumber, StateType aProposal) {
    ProposalFeedbackKind aFeedbackKind = ProposalFeedbackKind.SUCCESS;
    recordSentLearnNotification(aProposalNumber, aProposal, aFeedbackKind);
    sendLearnNotification(aProposalNumber, aProposal, aFeedbackKind);
}
```

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

Modify how to allow rejecting and better than asynchronous consistency?
### Replication Synchrony Extremes

- **Terminology:**
  - **D:** set of distributed processes/agents that need some level of replication synchrony
  - **O:** object to be replicated
  - **replica(d):** version of O kept by process d.
  - **value(p):** value proposed by proposal p
  - **commit(p, d):** time when replica(d) is assigned value(p).
  - **submit(p):** time when p is issued.
  - **state(d, t):** Value of replica(d) at time d
  - **proposer(p):** process that issues p.
  - **T:** set of all times after some replica starts

- **Asynchronous value replication:**
  - \( \text{commit}(p, d) - \text{submit}(p) \) is finite \( \forall d \in D \)

- **Pure synchronous (value) replication:**
  - \( \forall t \in T, \forall i, j \in D, \text{state}(i, t) = \text{state}(j, t) \)

- **Changing D?**
  - Assume process will eventually start and has checkpointed, so D can be a static set
Asynchronous Value Replication

Value  | State
--- | ---
42 | Consensus

1

propose(1, 42)

learn(1, 42)

2

42 | Consensus

3

42 | Consensus

4

42 | Consensus
Replication Synchrony Extremes (Review)

Terminology:
- \( D \): set of distributed processes/agents that need some level of replication synchrony
- \( O \): object to be replicated
- \( \text{replica}(d) \): version of \( O \) kept by process \( d \).
- \( \text{value}(p) \): value proposed by proposal \( p \)
- \( \text{commit}(p, d) \): time when \( \text{replica}(d) \) is assigned \( \text{value}(p) \).
- \( \text{submit}(p) \): time when \( p \) is issued.
- \( \text{state}(d, t) \): Value of \( \text{replica}(d) \) at time \( d \)
- \( \text{proposer}(p) \): process that issues \( p \).
- \( T \): set of all times after some replica starts

Asynchronous value replication:
- \( \text{commit}(p, d) - \text{submit}(p) \) is finite \( \forall d \in D \)

Pure synchronous (value) replication:
- \( \forall t \in T, \forall i, j \in D, \text{state}(i, t) = \text{state}(j, t) \)

Changing \( D \)?
- Assume process will eventually start and has checkpointed, so \( D \) can be a static set
Asynchronous Value Replication

Value  | State
--- | ---
42  | Consensus
42  | Consensus
42  | Consensus
42  | Consensus

Node 1:
- Value: 42
- State: Consensus
- Transaction: propose(1, 42)
- Transaction: learn(1, 42)

Node 2:
- Value: 42
- State: Consensus

Node 3:
- Value: 42
- State: Consensus

Node 4:
- Value: 42
- State: Consensus

Transactions flow from Node 1 to Nodes 2, 3, and 4.
SYNCHRONOUS REPLICATION ASSUMPTIONS

- Synchronous Distributed Systems
  - Global clock
  - Upper bound on successful message transmission time ($T^N$)
- Upper bound on when process receives message after arrival ($T^S$).
- $T = T^N + T^S$
SYNCHRONOUS VALUE REPLICATION

<table>
<thead>
<tr>
<th>Value</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Consensus</td>
</tr>
</tbody>
</table>

1. propose(1, 42)
2. Global clock unreasonable
3. Worst case T large
**Alternative Replication Synchrony**

- **Terminology:**
  - D: set of distributed processes/agents that need some level of replication synchrony
  - O: object to be replicated
  - Replica(d): version of O kept by process d.
  - Value(p): value proposed by proposal p
  - Commit(p, d): time when replica(d) is assigned value(p).
  - Submit(p): time when p is issued.
  - State(d, t): Value of Replica(d) at time d
  - Proposer(p): process that issues p.
  - T: set of all times when system is up

- **Asynchronous value replication:**
  \[ |\text{Commit}(p, \text{Proposer}(p) - \text{Commit}(p, d))| \text{ is finite } \forall d \in D \]

- **Pure Synchronous value replication:**
  \[ \forall t \in T, \forall i, j \in D, \text{State}(i, t) = \text{State}(j, t) \]

- **Synchronous (invalidation) replication**
  \[ \forall t \in T, \forall i, j \in D, \text{if State}(i, t) \neq \text{null}, \text{State}(j, t) = \text{null} | \text{State}(j, t) = \text{State}(i, t) \]

- **Protocol Change:**
  - Asynchronous: While message in transit, sending and receiving protocol at proposer not same
  - Requirement: Sending and receiving protocol be same
  - Idea: Invalidate protocol rather than pure synchronous
SYNCHRONOUS REPLICATION ASSUMPTIONS

- Synchronous Distributed Systems
  - Global clock
  - Upper bound on successful message transmission time

- Upper bound on when process receives message after arrival.
ONE-PHASE SYNCHRONOUS REPLICATION

Worst case T large

Safety vs. Progress/Liveness

Solution?

Value  State

42  Consensus

42  Consensus

42  Consensus

42  Consensus

propose(1, 42)

learn(2, 42)

T

T

T

T
ASYNCHRONOUS

Proposers

Learners
Synchronous

Proposers

Acceptors

Learners
**Two-Phase Synchronous Replication**

<table>
<thead>
<tr>
<th>Value</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Consensus</td>
</tr>
<tr>
<td>42</td>
<td>Consensus</td>
</tr>
<tr>
<td>42</td>
<td>Consensus</td>
</tr>
<tr>
<td>42</td>
<td>Consensus</td>
</tr>
</tbody>
</table>

- Value: 42
- State: Consensus

Propose(1, 42)
Accept(1, 42)
Learn(1, 42)
Send Accept Request to all Acceptors (instead of notification to all learners)

Acceptors will usually include self

Acceptors != learners?

if learners can be asynchronous (slaves vs mirrors)
1. **Propose(1, 42)**: This is the first step in the consensus process, where a value (42) is proposed.

2. **Accept(1, 42)**: The value is then accepted by the system.

3. **Invalidate State**: The system invalidates the current state to prepare for the next proposal.

4. **Send response to sender**: A response is sent to the sender acknowledging the acceptance of the value.

5. **Consensus** (Steps 2-4 are repeated with the same value): This step is repeated to ensure that all nodes in the system have accepted the same value, maintaining consensus across the network.
If not sufficient notifications record notification and return

Send learn notifications
LEADING-THINKING GENERAL PROBLEM

What if a receiving general vetoes (unable to process request)?

Problem compounded with N vetoing generals

Can take majority or allow veto power
## Feedback & Negative Learn

<table>
<thead>
<tr>
<th>Value</th>
<th>State</th>
<th>Value</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Consensus</td>
<td>0</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Consensus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Consensus</td>
</tr>
</tbody>
</table>

- Propose(1,42)
- Accept(1,42)
- Learn(1,42, no)
- Learn(1,42, yes)

**Consensus value is same even though 4 has not received invalidating accept request**

**Synchronous replication?**
**Semi-Synchronous Replication**

### Value | State
--- | ---
42 | Consensus
42 | Consensus
42 | Consensus
42 | Consensus

1. **propose(1,42)**
2. **accept(1,42)**
3. **learn(1,42, yes)**

- Consensus values different!
- Synchronous replication?
**Semi-Synchronous Replication**

- **Terminology:**
  - $D$: set of distributed processes/agents that need some level of replication synchrony
  - $O$: object to be replicated
  - $Replica(d)$: version of $O$ kept by process $d$
  - $Value(p)$: value proposed by proposal $p$
  - $Commit(p, d)$: time when replica($d$) is assigned value($p$).
  - $Submit(p)$: time when $p$ is issued.
  - $State(d, t)$: Value of Replica($d$) at time $d$
  - $Proposer(p)$: process that issues $p$.
  - $T$: set of all times when system is up

- **Asynchronous value replication:**
  - $|Commit(p, Proposer(p) – Commit(p, d))| \text{ is finite } \forall d \in D$

- **Pure synchronous value replication:**
  - $\forall t \in T, \forall i, j \in D, State(i, t) = State(j, t)$

- **Synchronous (invalidation) replication**
  - $\forall t \in T, \forall i, j \in D, \text{ if } State(i, t) \neq \text{ null, } State(j, t) = \text{ null } \mid State(j, t) = State(i, t)$

- **Semi-Synchronous (invalidation) replication**
  - $\forall t \in T, \forall i, j \in E, \text{ if } State(i, t) \neq \text{ null, } State(j, t) = \text{ null } \mid State(j, t) = State(i, t)$
  - $E \subseteq D, E \neq \emptyset$
Replication Synchrony

public enum ReplicationSynchrony {
    ASYNCHRONOUS,
    TWO_SYNCHRONOUS,
    MAJORITY_SYNCHRONOUS,
    ALL_SYNCHRONOUS,
    CUSTOM_SYNCHRONOUS
}

Majority in a dynamic set of acceptors?

public enum ConsensusMemberSetKind {
    CURRENT_MEMBERS,
    INITIAL_MEMBERS,
    MAXIMUM_MEMBERS
}
Accept

Invalidate State

Calculate verdict

Send verdict to sender

Value

State

42

Consensus

42

Consensus

42

Consensus

42

Consensus

propose(1,42)

accept(1,42)

learn(1,42, yes)
If proposal not pending return

If proposal member rejected and (all) synchronous replication, reject proposal and maybe send learn notification

If not sufficient notifications record notification and return

If proposal group rejected maybe send negative learn notifications and return

Send learn notifications
public void accept(float aProposalNumber, StateType aProposal) {
    recordReceivedAcceptRequest(aProposalNumber, aProposal);
    if (!isPending (aProposalNumber) && !isSendAcceptReplyForResolvedProposal()) {
        return;
    }
    ProposalFeedbackKind aFeedbackKind =
        checkAcceptRequest(aProposalNumber, aProposal);
    if (!isSuccess(aFeedbackKind) && !isSendRejectionNotification()) {
        return;
    }
    recordAndSendAcceptedNotification(aProposalNumber, aProposal, aFeedbackKind);
}

Why not return value?

Do not want to tie to RPC

Caller may not wait for response

Response may be sent to others than proposal sender
protected void localPropose(float aProposalNumber, StateType aProposal) {
    if (isAsynchronousConsistency()) {
        sendLearnNotification(aProposalNumber, aProposal, ProposalFeedbackKind.SUCCESS);
    } else {
        sendAcceptRequest(aProposalNumber, aProposal);
    }
}
public void accepted(float aProposalNumber, StateType aProposal, ProposalFeedbackKind aFeedbackKind) {
    ProposalAcceptResponseReceived.newCase(this, getObjectName(), ProposalNumber, aProposal, aFeedbackKind);
    recordReceivedAcceptedNotification(aProposalNumber, aProposal, aFeedbackKind);
    if (!isPending(aProposalNumber)) {
        return;
    }
    if (!isSuccess(aFeedbackKind) && isAllSynchronous()) {
        processAcceptRejection(aProposalNumber, aProposal, aFeedbackKind);
        return;
    }
    aggregateAcceptedNotification(
        aProposalNumber, aProposal, aFeedbackKind);
}
protected void aggregateAcceptedNotification(
    float aProposalNumber, StateType aProposal,
    ProposalFeedbackKind aFeedbackKind) {
    Boolean isSufficientAcceptors = sufficientAcceptors(
        aProposalNumber, aProposal);
    if (isSufficientAcceptors == null)
        return;
    if (isSufficientAcceptors) {
        recordAndSendLearnNotification(aProposalNumber,
            aProposal, ProposalFeedbackKind.SUCCESS);
    } else {
        processAcceptRejection(aProposalNumber, aProposal,
            ProposalFeedbackKind.AGGREGATE_DENIAL);
    }
}
PRACTICAL APPLICATION WITH REJECTION

Want to know if file has reached server before shutting down laptop

Want to know if file actually savable by server

Want conflicts to not cause inconsistent files

Want to know if file has reached server before shutting down laptop

Reason for not accepting?
PRÁCTICAL REASON FOR REJECTING
**Feedback Kind and Proposal State**

```java
public enum ProposalFeedbackKind {
    SUCCESS,
    SERVICE_FAULT,
    ACCESS_DENIAL,
    SERVICE_DENIAL,
    CONCURRENT_OPERATION,
    AGGREGATE_DENIAL // consolidated response
}
```

Checks made by application vs consensus abstraction?

Abstraction

```java
public enum ProposalState {
    PROPOSAL_CONSENSUS,
    PROPOSAL_PENDING,
    PROPOSAL_SERVICE_FAULT,
    PROPOSAL_ACCESS_DENIAL,
    PROPOSAL_SERVICE_DENIAL,
    PROPOSAL_CONCURRENT_OPERATION,
    PROPOSAL_AGGREGATE_DENIAL
}
```

How does application get involved?
**VETOER DESIGN PATTERN**

Consensus Mechanism <StateType>

```java
void addConsensusListener(ConsensusListener<StateType> aConsensusListener);

void addConsensusVetoer(ProposalVetoer<StateType> aConsensusVetoer);
```
public class AMeaningOfLifeVetoer implements ProposalVetoer<Integer>{
    public ProposalFeedbackKind acceptProposal(float aProposalNumber, Integer aState) {
        return aState % 2 != 0?
            ProposalFeedbackKind.SERVICE_DENIAL:ProposalFeedbackKind.SUCCESS;
    }
}

protected synchronized ProposalFeedbackKind checkProposalForAccept(
    float aProposalNumber, StateType aProposal ) {
    return checkWithVetoer(aProposalNumber, aProposal);
}

protected synchronized ProposalFeedbackKind checkWithVetoer(
    float aProposalNumber, StateType aProposal ) {
    for (ProposalVetoer<StateType> aConsensusVetoer:consensusVetoers){
        ProposalFeedbackKind aRejectionKind =
            aConsensusVetoer.acceptProposal(aProposalNumber, aProposal);
        if (aRejectionKind != ProposalFeedbackKind.SUCCESS) {
            return aRejectionKind;
        }
    }
    return ProposalFeedbackKind.SUCCESS;
}
FOLLOWING ACCEPTOR 3 (SYNCHRONOUS)

I**(ProposalAcceptRequestReceived)  Meaning, 1.0002=29
I**(ProposalAcceptedNotificationSent)  Meaning, 1.0002=29 --> SUCCESS
I**(ProposalAcceptRequestReceived)  Meaning, 1.0001=42
I**(ProposalAcceptedNotificationSent)  Meaning, 1.0001=42 --> SUCCESS
I**(ProposalLearnNotificationReceived)  Meaning, 1.0001=42 --> SUCCESS
I**(ProposalStateChanged)  Meaning, 1.0001=42 --> PROPOSAL_CONSENSUS
Meaning of Life: 42
I**(ProposalLearnNotificationReceived)  Meaning, 1.0002=29 --> SERVICE_DENIAL
I**(ProposalStateChanged)  Meaning, 1.0002=29 --> PROPOSAL_SERVICE_DENIAL
THINKING ACCEPTOR 4 (SYNCHRONOUS)

I*** (ProposalAcceptRequestReceived)  Meaning, 1.0001=42
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0001=42-->SUCCESS
I*** (ProposalAcceptRequestReceived)  Meaning, 1.0002=29
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0002=29-->SERVICE_DENIAL
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42-->SUCCESS
I*** (ProposalStateChanged) Meaning, 1.0001=42-->PROPOSAL_CONSENSUS
Meaning of Life: 42
I*** (ProposalLearnNotificationReceived) Meaning, 1.0002=29-->SERVICE_DENIAL
I*** (ProposalStateChanged) Meaning, 1.0002=29-->PROPOSAL_SERVICE_DENIAL
### Accepted Proposer 1 (Synchronous)

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProposalMade</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptRequestSent</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalWaitStarted</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptRequestReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptedNotificationSent</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptRequestReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptedNotificationSent</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalAcceptResponseReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>SufficientAgreementsChecked</td>
<td>1.0001=42:1</td>
</tr>
<tr>
<td>ProposalAcceptResponseReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>SufficientAgreementsChecked</td>
<td>1.0001=42:2</td>
</tr>
<tr>
<td>ProposalAcceptResponseReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>SufficientAgreementsChecked</td>
<td>1.0001=42:3</td>
</tr>
<tr>
<td>ProposalAcceptResponseReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>SufficientAgreementsChecked</td>
<td>1.0001=42:4</td>
</tr>
<tr>
<td>ProposalLearnNotificationSent</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalLearnNotificationReceived</td>
<td>1.0001=29--SERVICE_DENIAL</td>
</tr>
<tr>
<td>ProposalStateChanged</td>
<td>1.0002=29--PROPOSAL_STATE_CHANGED</td>
</tr>
<tr>
<td>ProposalLearnNotificationReceived</td>
<td>1.0001=42</td>
</tr>
<tr>
<td>ProposalStateChanged</td>
<td>1.0001=42--PROPOSAL_CONSENSUS</td>
</tr>
<tr>
<td>Meaning of Life</td>
<td>42</td>
</tr>
<tr>
<td>ProposalWaitEnded</td>
<td>1.0001=42--PROPOSAL_CONSENSUS</td>
</tr>
</tbody>
</table>
REJECTED PROPOSER 2 (SYNCHRONOUS)

I*** (ProposalMade) Meaning, 1.0002=29
I*** (ProposalAcceptRequestSent) Meaning, 1.0002=29
I*** (ProposalWaitStarted) Meaning, 1.0002=29
I*** (ProposalAcceptRequestReceived) Meaning, 1.0001=42
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:1|1|4|4?4.0 --> null
I*** (ProposalAcceptRequestReceived) Meaning, 1.0002=29
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:2|2|4|4?4.0 --> null
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --&gt; SUCCESS
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:3|3|4|4?4.0 --> null
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --&gt; SERVICE_DENIAL
I*** (ProposalLearnNotificationSent) Meaning, 1.0002=29 --&gt; SERVICE_DENIAL
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42 --&gt; SUCCESS
I*** (ProposalStateChanged) Meaning, 1.0001=42 --&gt; PROPOSAL_CONSENSUS
Meaning of Life: 42
I*** (ProposalLearnNotificationReceived) Meaning, 1.0002=29 --&gt; SERVICE_DENIAL
I*** (ProposalStateChanged) Meaning, 1.0002=29 --&gt; PROPOSAL_SERVICE_DENIAL
I*** (ProposalWaitEnded) Meaning, 1.0002=29 --&gt; PROPOSAL_SERVICE_DENIAL
Accepted Proposer 2 (Two-Synchronous)

I*** (ProposalMade) Meaning, 1.0002=29
I*** (ProposalAcceptRequestSent) Meaning, 1.0002=29
I*** (ProposalWaitStarted) Meaning, 1.0002=29
I*** (ProposalAcceptRequestReceived) Meaning, 1.0001=42
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:1|1|4|4?2.0 --> null
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SERVICE_DENIAL
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:1|2|4|4?2.0 --> null
I*** (ProposalAcceptRequestReceived) Meaning, 1.0002=29
I*** (ProposalAcceptedNotificationSent) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (SufficientAgreementsChecked) Meaning, 1.0002, 29:2|3|4|4?2.0 --> true
I*** (ProposalLearnNotificationSent) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalLearnNotificationReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalStateChanged) Meaning, 1.0002=29 --> PROPOSAL_CONSENSUS
Meaning of Life: 29
I*** (ProposalWaitEnded) Meaning, 1.0002=29 --> PROPOSAL_CONSENSUS
I*** (ProposalAcceptResponseReceived) Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalStateChanged) Meaning, 1.0001=42 --> PROPOSAL_CONSENSUS
Meaning of Life: 42
I*** (ProposalLearnNotificationReceived) Meaning, 1.0001=42 --> SUCCESS
Accepted Proposer 1 (Two-Synchronous)

Proposal state changed when learn invocation is later scheduled
Accepted Proposer 1 (Two-Synchronous, Immediate State Change)

I*** (ProposalMade)  Meaning, 1.0001=42
I*** (ProposalAcceptRequestSent)  Meaning, 1.0001=42
I*** (ProposalWaitStarted)  Meaning, 1.0001=42
I*** (ProposalAcceptRequestReceived)  Meaning, 1.0002=29
I*** (ProposalAcceptedNotificationSent)  Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalAcceptResponseReceived)  Meaning, 1.0001=42 --> SUCCESS
I*** (SufficientAgreementsChecked)  Meaning, 1.0001, 42:1|1|4|4?2.0 --> null
I*** (ProposalAcceptResponseReceived)  Meaning, 1.0001=42 --> SUCCESS
I*** (SufficientAgreementsChecked)  Meaning, 1.0001, 42:2|2|4|4?2.0 --> true
I*** (ProposalLearnNotificationSent)  Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalLearnNotificationReceived)  Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalStateChanged)  Meaning, 1.0001=42 --> PROPOSAL_CONSENSUS
Meaning of Life: 42
I*** (ProposalWaitEnded)  Meaning, 1.0001=42 --> PROPOSAL_CONSENSUS
I*** (ProposalAcceptRequestReceived)  Meaning, 1.0001=42
I*** (ProposalAcceptedNotificationSent)  Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalAcceptResponseReceived)  Meaning, 1.0001=42 --> SUCCESS
I*** (ProposalLearnNotificationReceived)  Meaning, 1.0002=29 --> SUCCESS
I*** (ProposalStateChanged)  Meaning, 1.0002=29 --> PROPOSAL_CONSENSUS
Meaning of Life: 29
I*** (ProposalAcceptResponseReceived)  Meaning, 1.0001=42 --> SUCCESS
(Semi) Synchronous 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 ❌