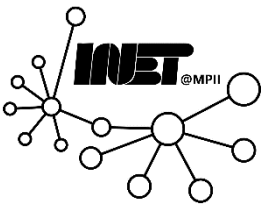
Data Networks
State

Prof. Anja Feldmann, Ph.D.



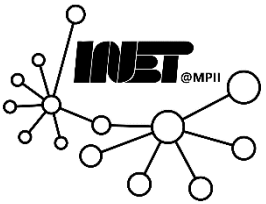
Design Principles

Goals:

- Identify, study common architectural components, protocol mechanisms, approaches do we find in network architectures?
- Synthesis: Big picture

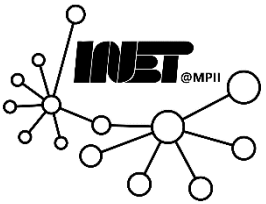
Design Principles:

- Separation of data, control
- Hard state versus soft state
- Randomization
- Indirection
- Network virtualization / Overlays
- Resource sharing
- Design for scale



1: Separation of control and data

- PSTN (public switched telephone network):
  - SS7 (packets-switched control network) separate from (circuit-switched) call trunk lines
  - Earlier tone-based (in-band signaling)
- Internet:
  - HTTP: in-band signaling
  - FTP: out-of-band signaling
  - RSVP (signaling) separate from routing, forwarding.



Internet: HTTP - inband signaling

Suppose user enters the following URL

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 JPEG images)

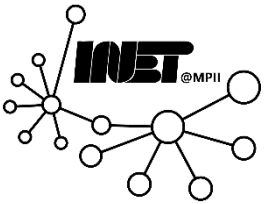
1a. HTTP client initiates TCP connection to HTTP server (process) at *www.someSchool.edu* on port *80*

1b. HTTP server at host *www.someSchool.edu* waiting for TCP connection at port 80. "accepts" connection, notifying client

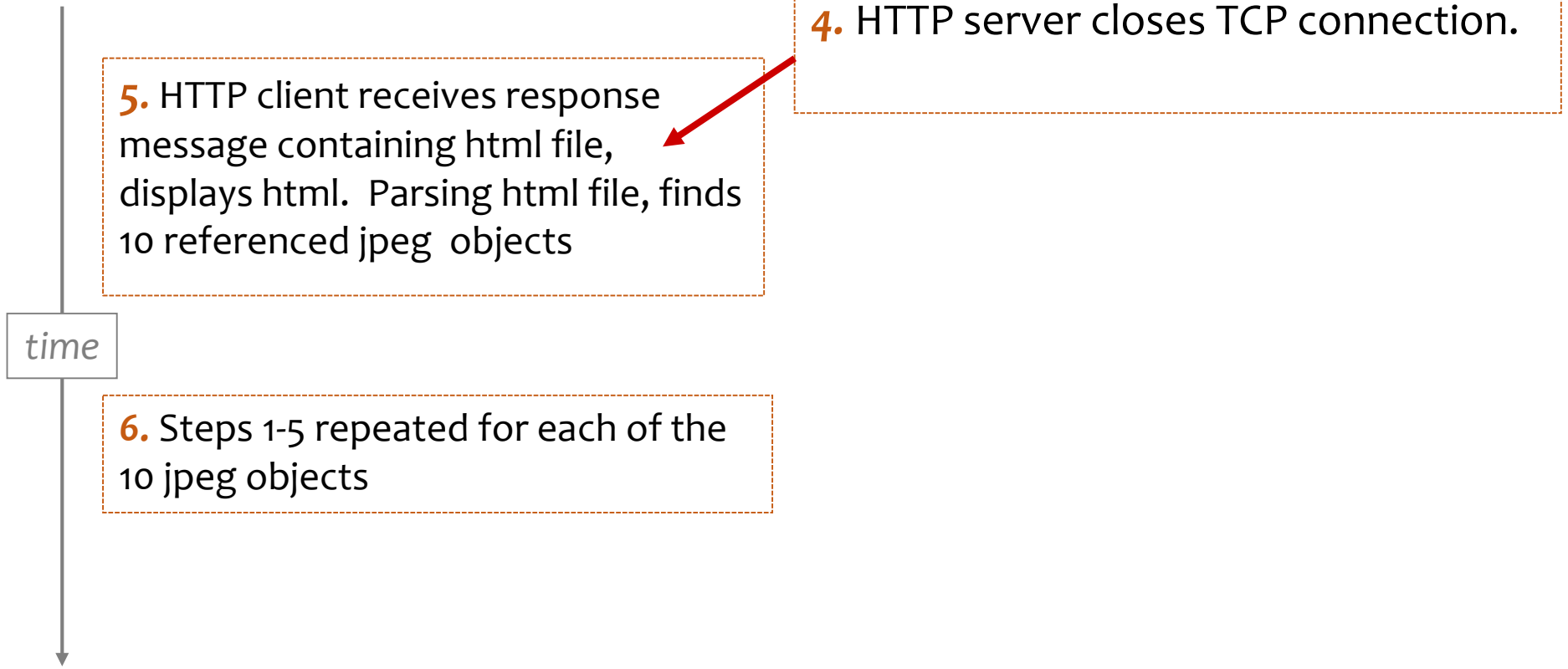
time

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object *someDepartment/home.index*

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket



Internet: HTTP - inband signaling (2)



HTTP request message

request line (GET, POST, HEAD commands)

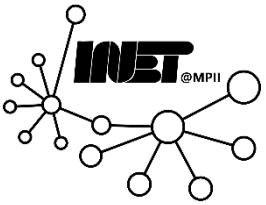
header lines

Carriage return, line feed indicates end of message

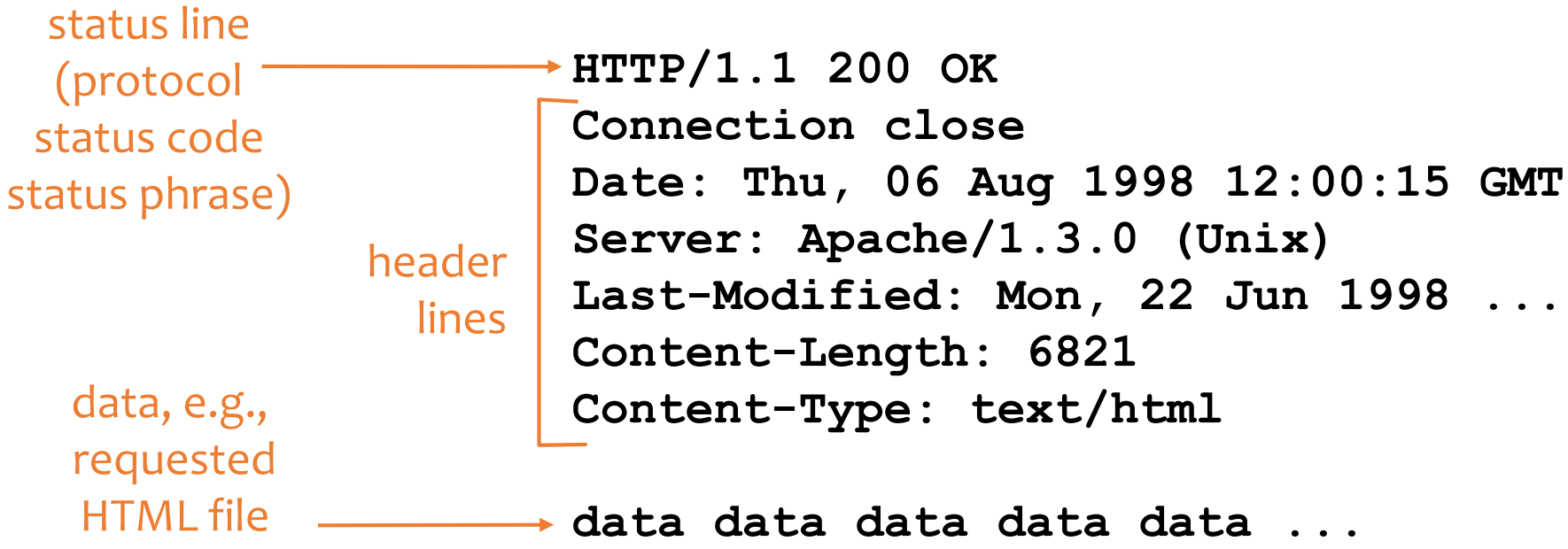
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

Note: Request msg typically just a signaling msg (no data)



HTTP response message

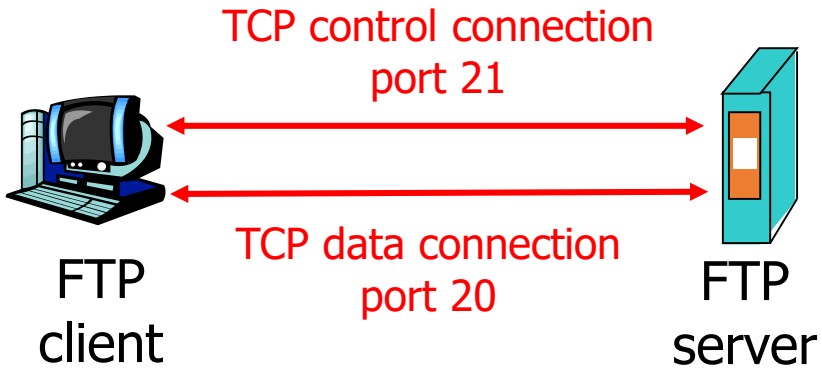


Note: Response msg mixes signaling and data

- Request, response msgs exchanged over *single* TCP connection

FTP: Separate control, data connections

- FTP client contacts FTP server at port 21
- Client obtains authorization over control connection
- Client browses remote directory via commands sent over control connection
- When server receives file transfer command server opens new TCP data connection to client
- After transferring one file, server closes connection



- Server opens new TCP data connection to transfer another file
- Control connection:
  "Out of band" signaling
- FTP server maintains "state":
  Current directory, earlier authentication

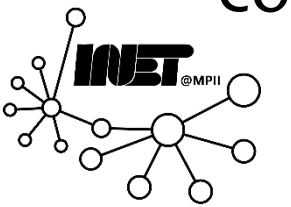
Separate control, data: Why (or why not)?

Why?

- Allows concurrent control + data
- Allows perform authentication at control level
- Simplifies processing of data/control streams – higher throughput
- Provide QoS appropriate for control/data streams

Why not?

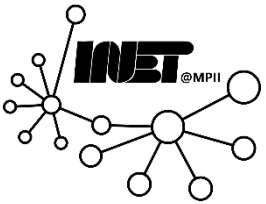
- Separate channels complicate management, increases resource requirements
- Can increase latency, e.g., http – two top connections vs. one



2: Maintaining network state

State: Information *stored* in network nodes by network protocols

- Updated when network "conditions" change
- Stored in multiple nodes
- Often associated with end-system generated call or session
- Examples:
  - TCP: Sequence numbers, timer values, RTT estimates
  - RSVP: Router maintain lists of
    - Upstream sender IDs
    - Downstream receiver reservations



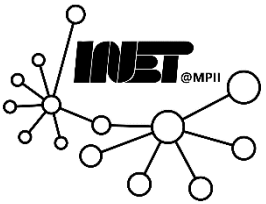
State: Sender, receiver

- Sender: Network node that (re)generates signaling (control) msgs to install, keep-alive, remove state from other nodes
- Receiver: Node that creates, maintains, removes state based on signaling msgs received from sender

Hard-state

- State installed by receiver via setup msg from sender
- State removed by receiver via teardown msg from sender

- Default assumption: State valid unless told otherwise
  - In practice: Failsafe-mechanisms (to remove orphaned state)
    E.g., receiver-to-sender "heartbeat": Is this state still valid?
- Examples:
  - TCP

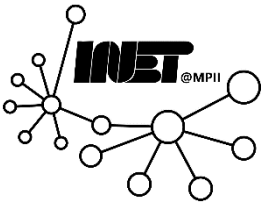


Soft-state

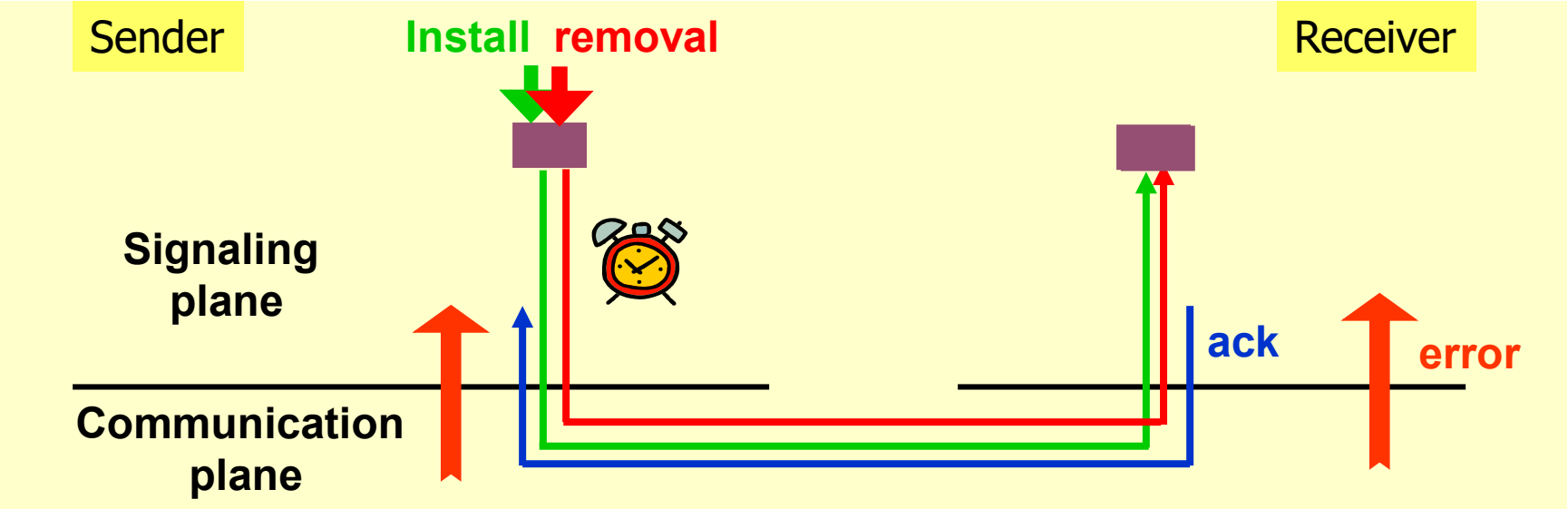
- State installed by receiver via setup (trigger) msg from sender (typically, an endpoint)

  Sender sends periodic refresh msg indicating receiver should continue to maintain state
- State removed by receiver via timeout (absence of refresh msg from sender)

- Default assumption: State becomes invalid unless refreshed
  - In practice: Explicit state removal (teardown) msgs may also be used
- Examples:
  - RSVP, RTP, IGMP



Hard-state signaling



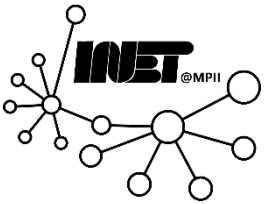
- Reliable signaling
- State removal by request
- Requires additional error handling
  - E.g., Sender failure

Soft-state

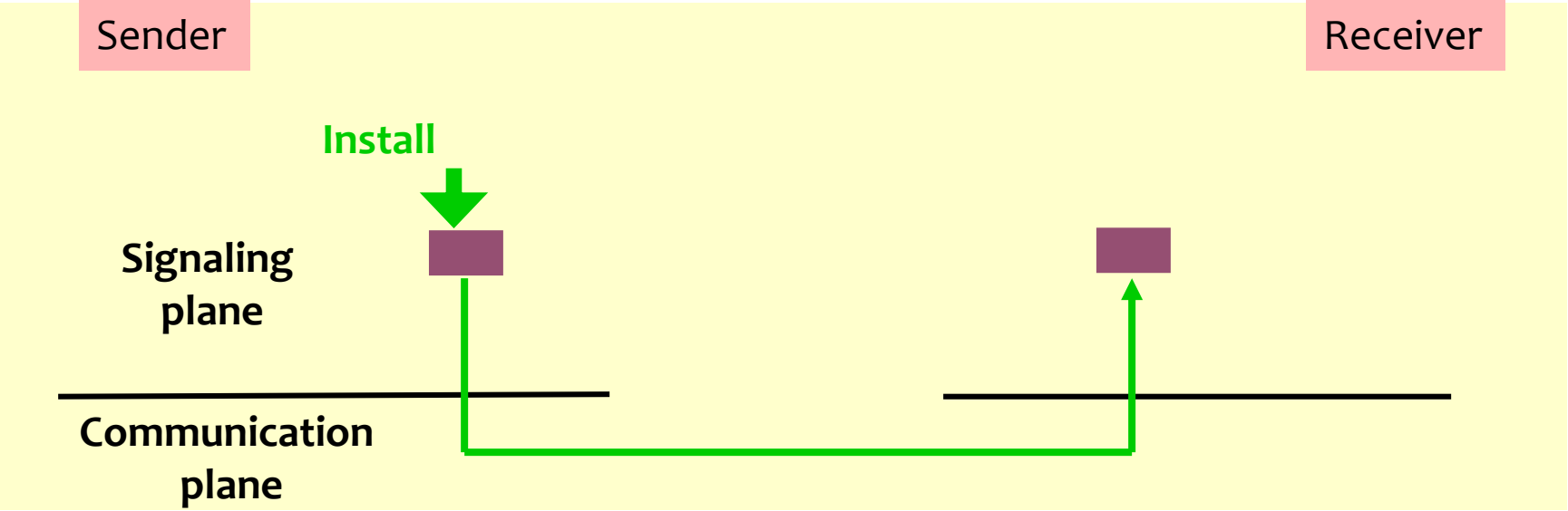
- State installed by receiver via setup (trigger) msg from sender (typically, an endpoint)

  Sender sends periodic refresh msg indicating receiver should continue to maintain state
- State removed by receiver via timeout (absence of refresh msg from sender)

- Default assumption: State becomes invalid unless refreshed
  - In practice: Explicit state removal (teardown) msgs may also be used
- Examples:
  - RSVP, RTP, IGMP

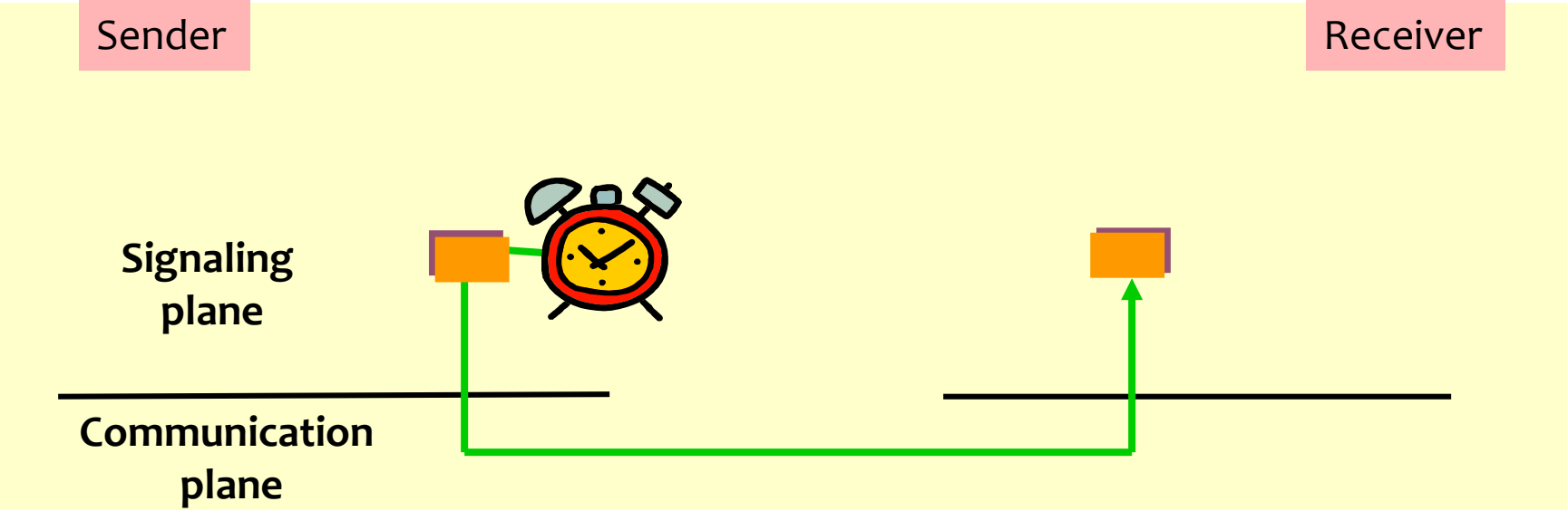


Soft-state signaling



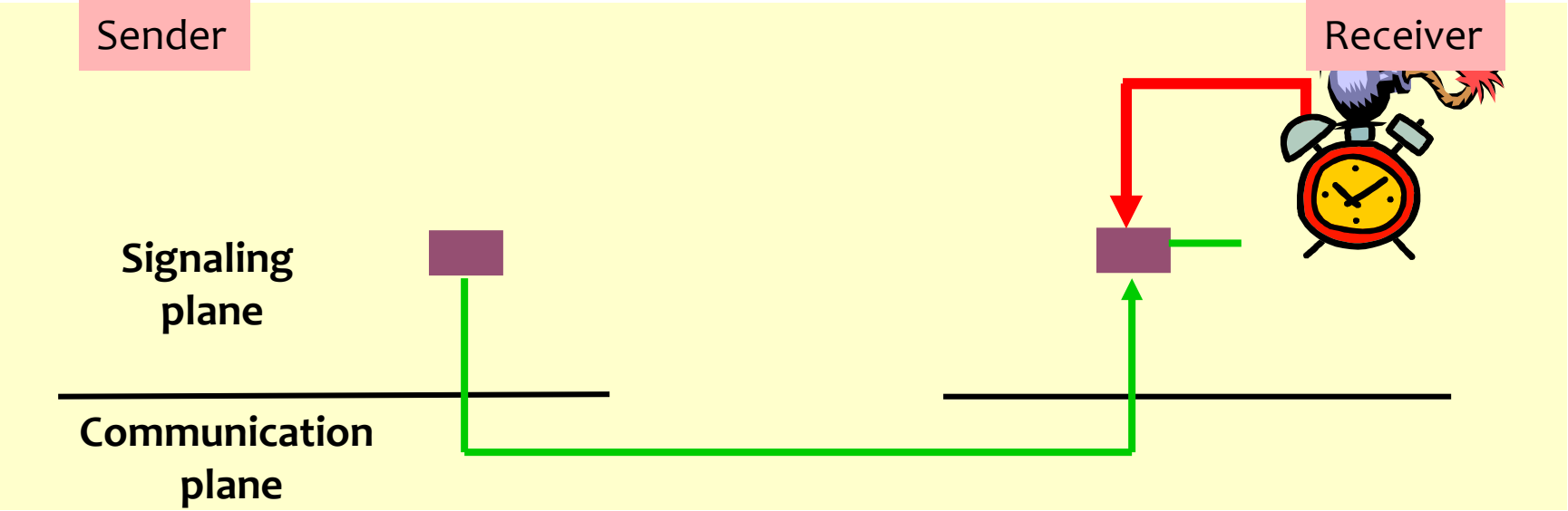
- Best effort signaling

Soft-state signaling



- Best effort signaling
- Refresh timer, periodic refresh

Soft-state signaling

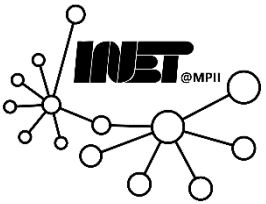


- Best effort signaling
- Refresh timer, periodic refresh
- State time-out timer, state removal only by time-out

Soft-state: Claims

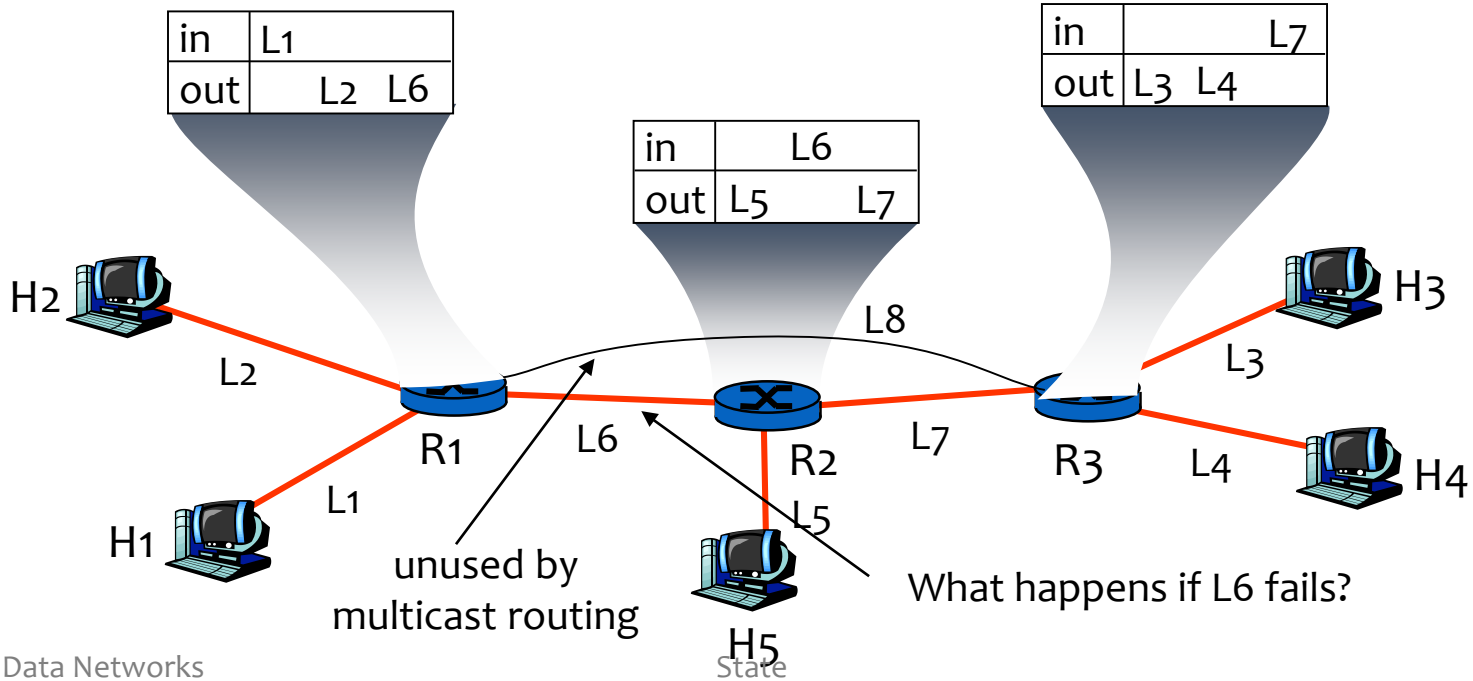
- "Systems built on soft-state are robust" [Raman 99]
- "Soft-state protocols provide … greater robustness to changes in the underlying network conditions …" [Sharma 97]
- "Obviates the need for complex error handling software" [Balakrishnan 99]

What does this mean?



Soft-state: "Easy" handling of changes

- Periodic refresh: If network "conditions" change, refresh will re-establish state under new conditions
- Example: RSVP/routing interaction: If routes change (nodes fail) RSVP PATH refresh will re-establish state along new path



Soft-state: "Easy" handling of changes

- "Recovery" performed transparently to end-system by normal refresh procedures
- No need for network to signal failure/change to end system, or end system to respond to specific error
- Less signaling (volume, types of messages) than hard-state from network to end-system but…
- More signaling (volume) than hard-state from end-system to network for refreshes

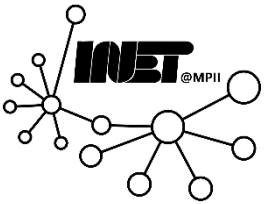
Soft-state: Refreshes

Refresh msgs serve many purposes:

- Trigger: first time state-installation
- Refresh: refresh state known to exist ("I am still here")
- <Lack of refresh>: Remove state ("I am gone")

Challenge: All refresh msgs unreliable

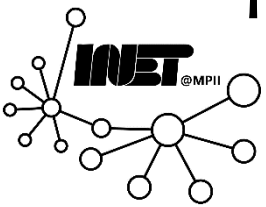
- Would like triggers to result in state-installation asap
- Enhancement: Add receiver-to-sender refresh_ACK for triggers
- E.g., see "Staged Refresh Timers for RSVP"



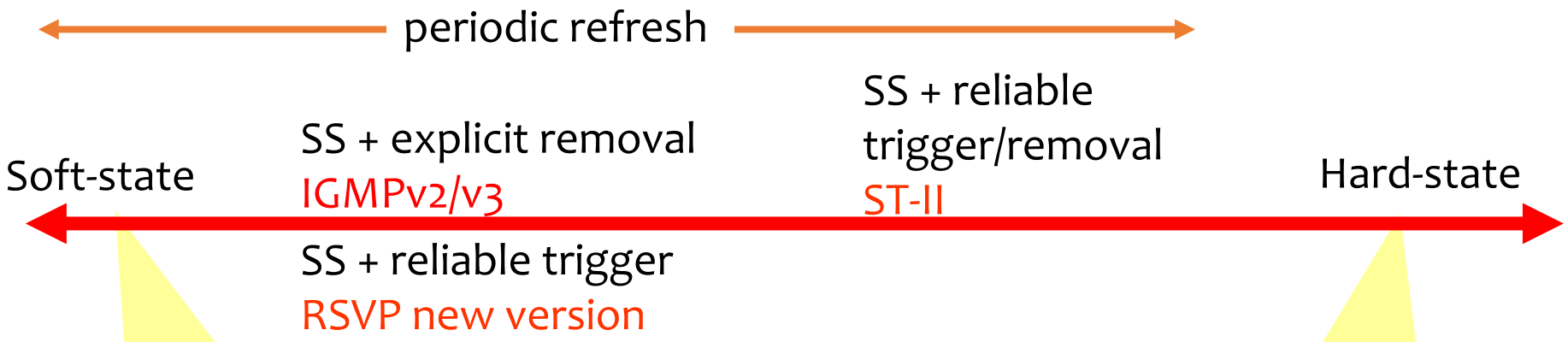
Soft-state: Setting timer values

Q: How to set refresh/timeout timers

- State-timeout interval = n * refresh-interval-timeout
  - What value of n to choose?
- Will determine amount of signaling traffic, responsiveness to change
  - Small timers: Fast response to changes, more signaling
  - Long timers: Slow response to changes, less signaling
- Ultimately: Consequence of slow/fast response, msg loss probability will dictate appropriate timer values



Signaling spectrum



- Best effort periodic state installation/refresh
- State removal by time out
- RSVP, IGMPv1

- Reliable signaling
- Explicit state removal
- Requires additional mechanism to remove orphan state
- SS7, TCP

Hard-state versus soft-state: Discussion

Q: Which is preferable and why?