Federated Systems

Lecture 4
Network Interfaces
Richard Martin
Sept. 28th 1999

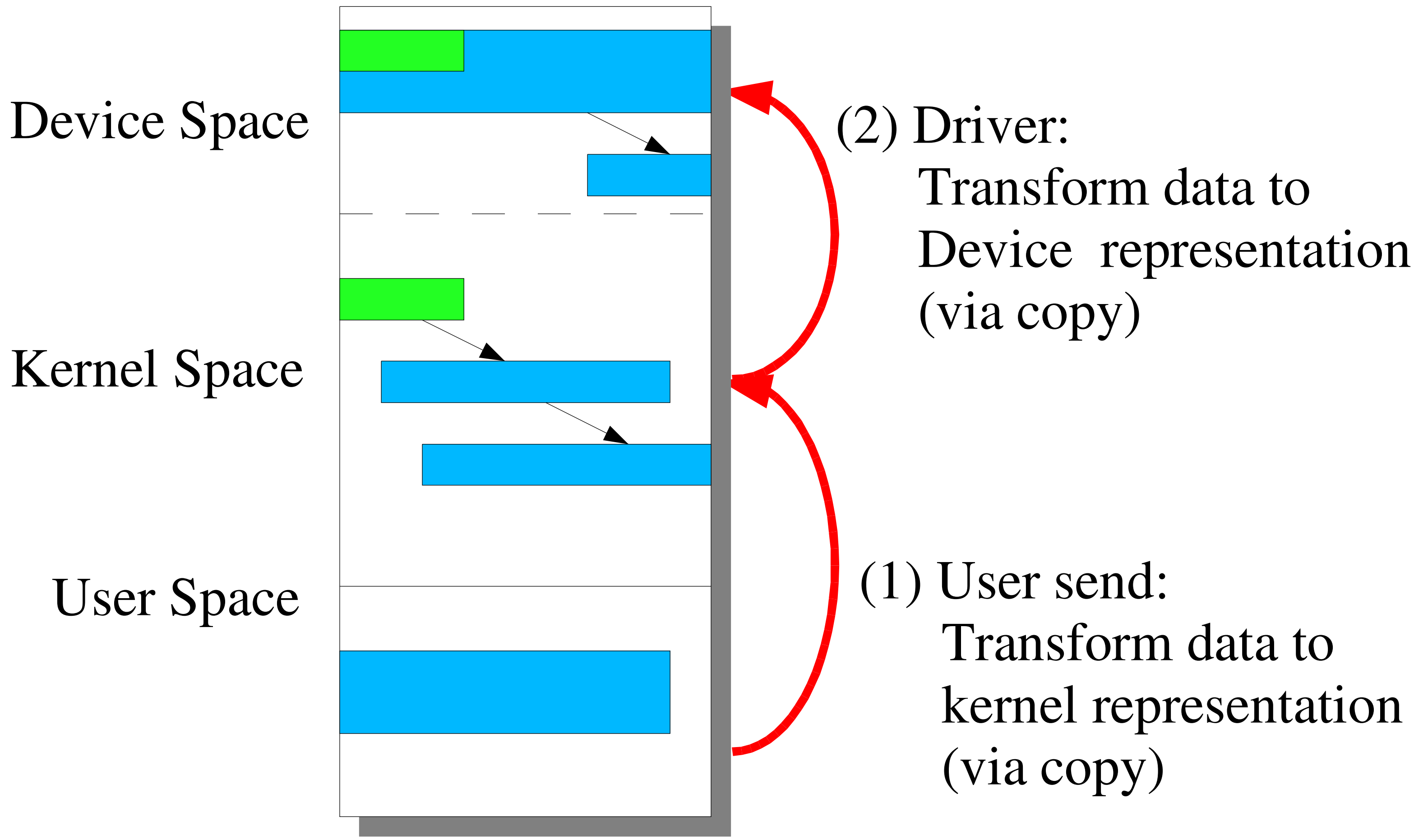
Today

- Quick overview of Network Interfaces
    - AfterBurner
    - VIA
    - Shrimp
- Issues
    - Data and Control Placement
    - Performance
    - Semantics

Today

- Why are we looking at these interfaces?
- Understand how data gets moved around
- Understand programming interfaces
- Quantify trade–offs of different architectures
  - Performance
  - Semantics
- Helps structure applications

Typical Kernel Architecture

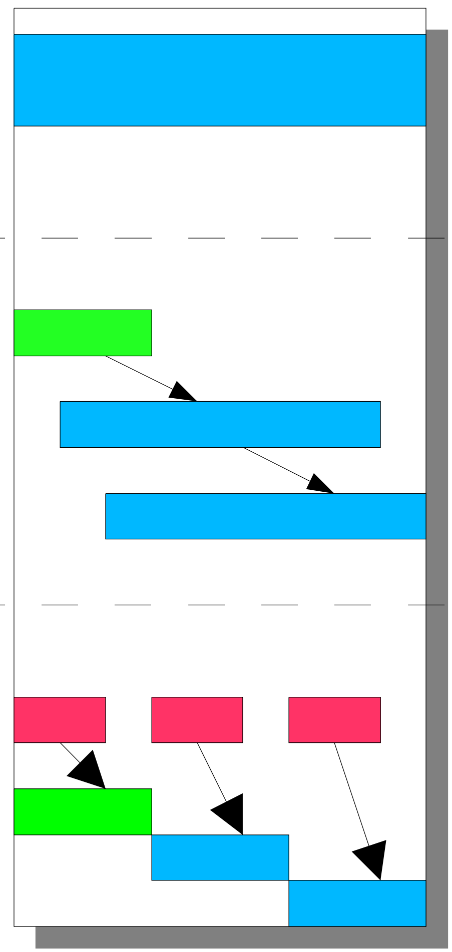


Increasing Performance

- Network access w/o all those layers?
- Map in NI to user process
- Fast, but what about sharing?
    - Multiple physical devices (BIP, FM, AM1)
    - Swap entire sets of devices (PM, HPAM)
    - Memory–mapped abstraction (Shrimp)
    - Virtualized device layer (VIA)

Historic Confusion

User Space



User's representation

Kernel Space

Kernel's representation of the message

Device Space

Device's representation of the message

What’s important

- Protocol not important
  - "TCP is too slow" misses the point
- Real issue is that there are too many transformations between
  - Representations
  - Address Spaces (logical copies)
  - Threads of Control (User/Kernel/Device)

Fundamental Tradeoffs

- No free lunch....
- Availability vs. Performance
  - requires 2nd logical thread of control...
- Performance vs. Functionality
  - Not just semantics of protocol
  - Reach of layer? Who/what can you talk to?
- Manageability vs. Performance
  - What takes care of set–up/tear down, routing, etc.

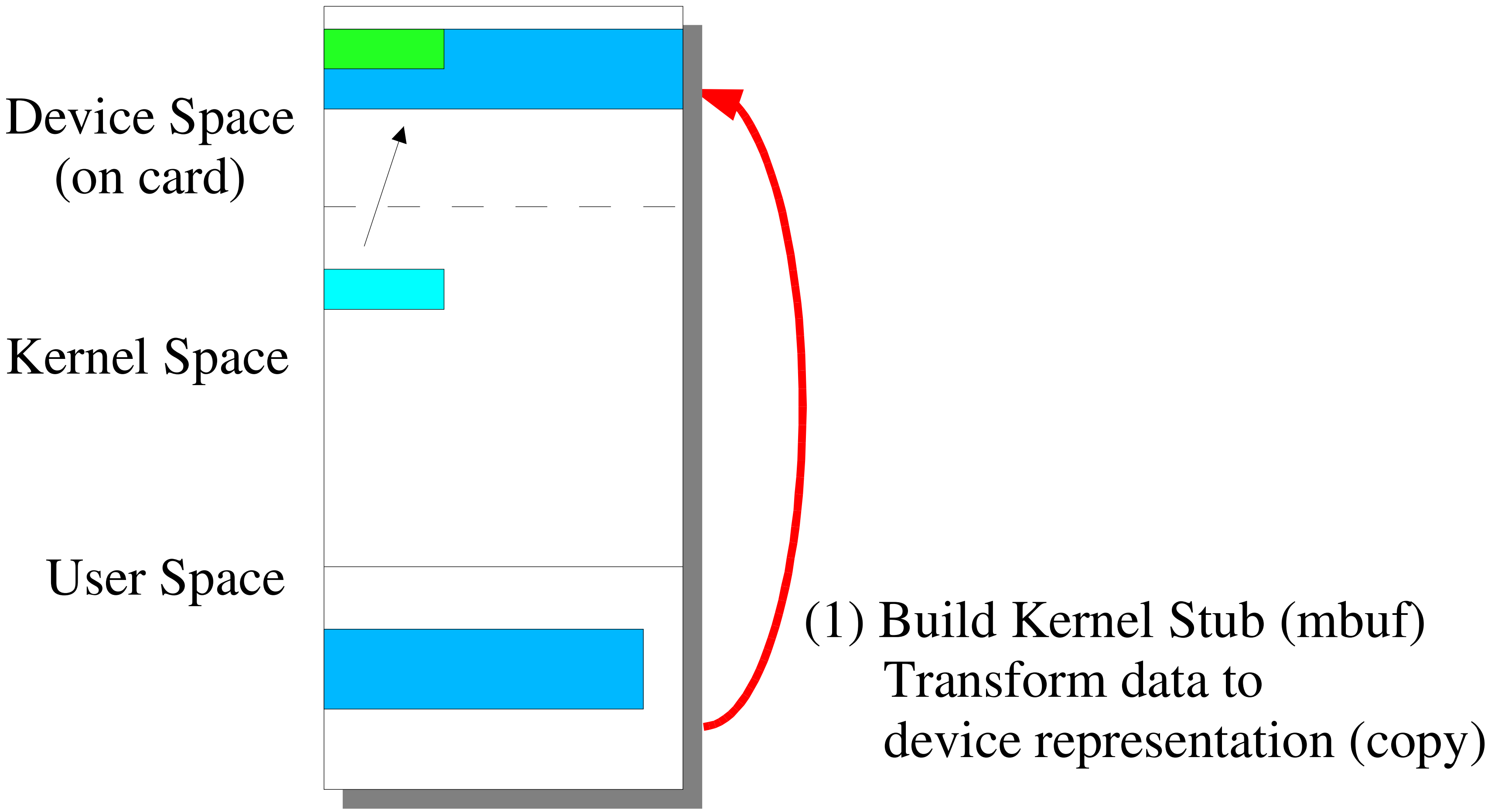
Solving performance problems

- Afterburner
  - Single copy provides a way to do everything at once.
    - Collapse all transformations into 1 operation
    - F(G(H(X))) –> C(X)
- VIA
  - provide 1 layer, punt to application
- Shrimp/Reflective Memory
  - Not even Q semantics, just move data

Afterburner

- Problem: Copying Data
  - Copying = bad performance?
  - Why Copy?
    - Semantics
      - Semantics: re–use buffer post send
      - "Functional layering"; user/kernel/device boundaries
    - Performance
      - Focus = bulk Gap (and o vs. G)
  - How important is G? vs. o or g?

Single Copy


Device Space
(on card)
Kernel Space
User Space
(1) Build Kernel Stub (mbuf)
Transform data to
device representation (copy)

Afterburner Copy Reduction

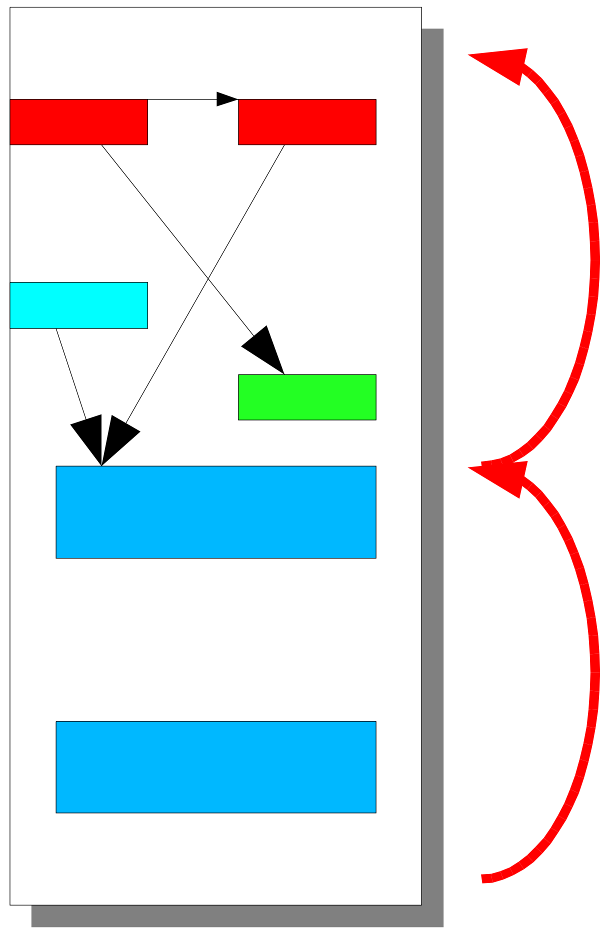
- Reduce representation overhead?
- Medusa/Afterburner idea:
  - Move data to card memory on first transformation
- Requires changes to kernel representation
- Add support for IP checksum (16 bit OR)

Copy on Write

Device Space?

Kernel Space

User Space



(2) Device Driver
Point Device structs to
headers + data

(1) User send:
Update page tables
Build Kernel structures

Hardware FIFOs

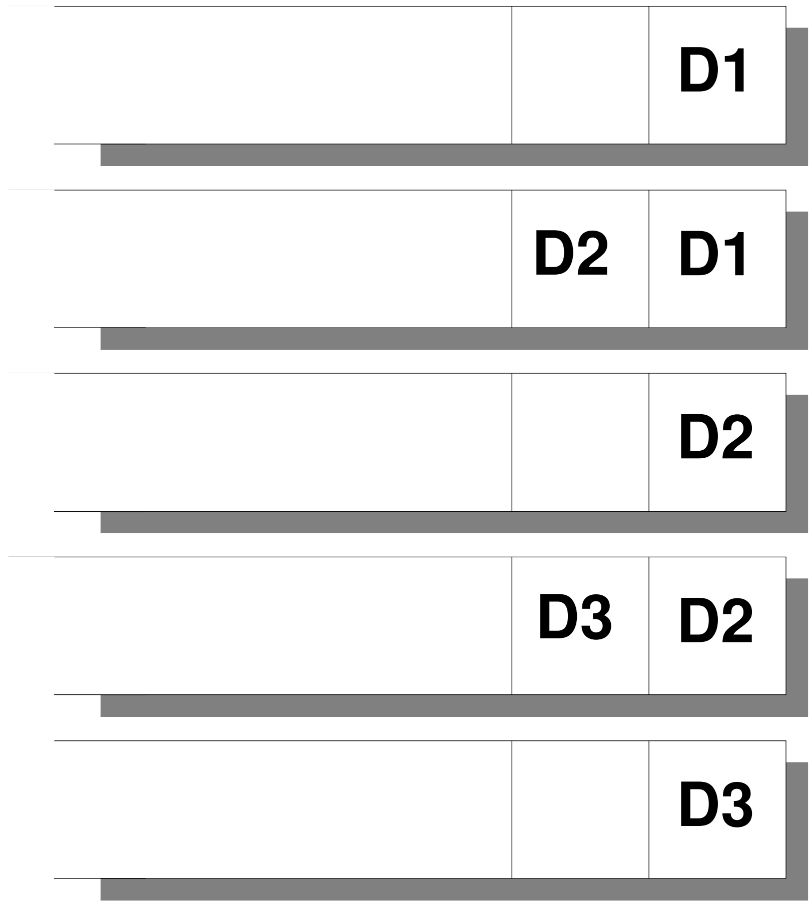
Store D1 at A1

Store D2 at A1

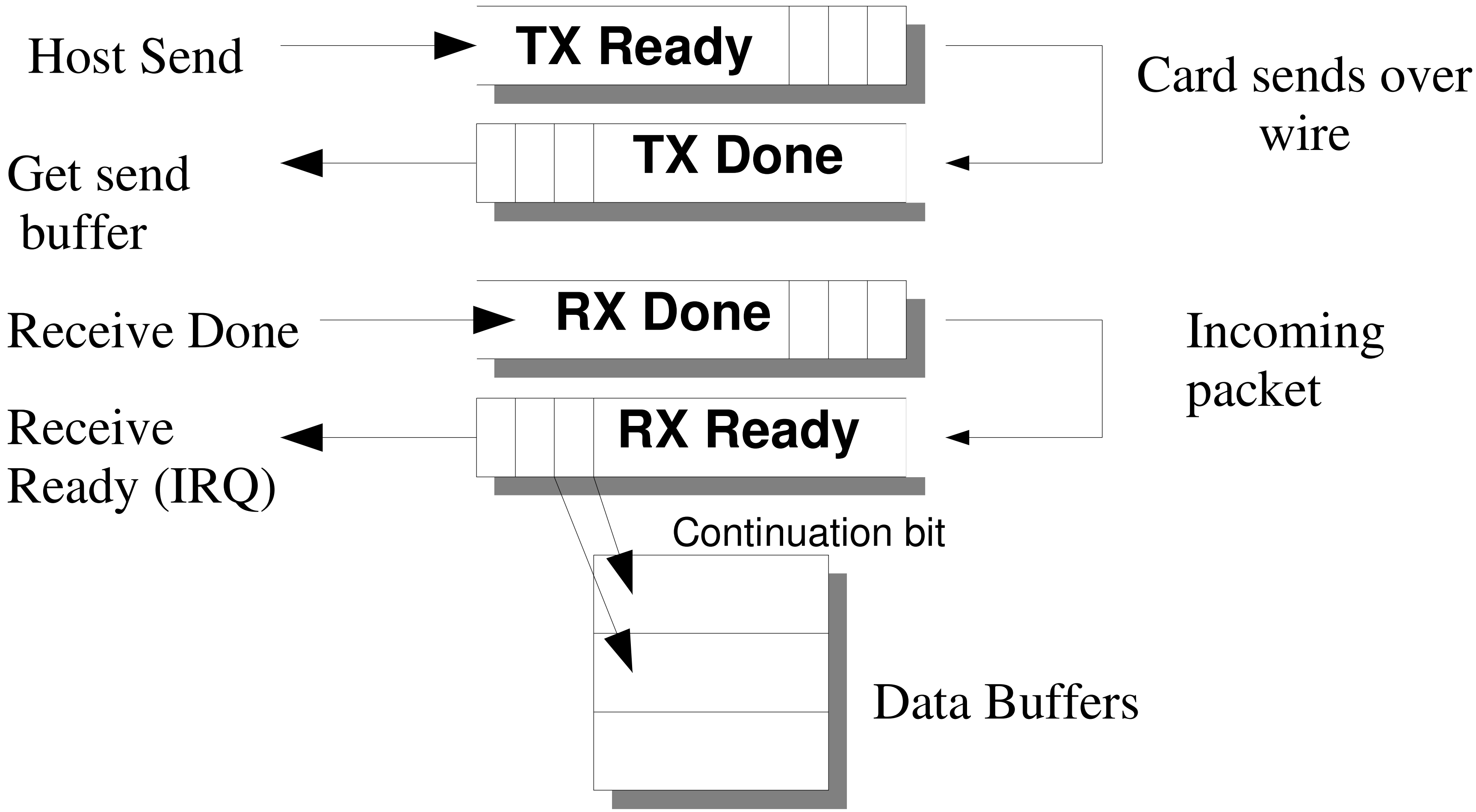
D1 = Load at A1

Store D3 at A1

D2 = Load at A1



Afterburner Data Structures

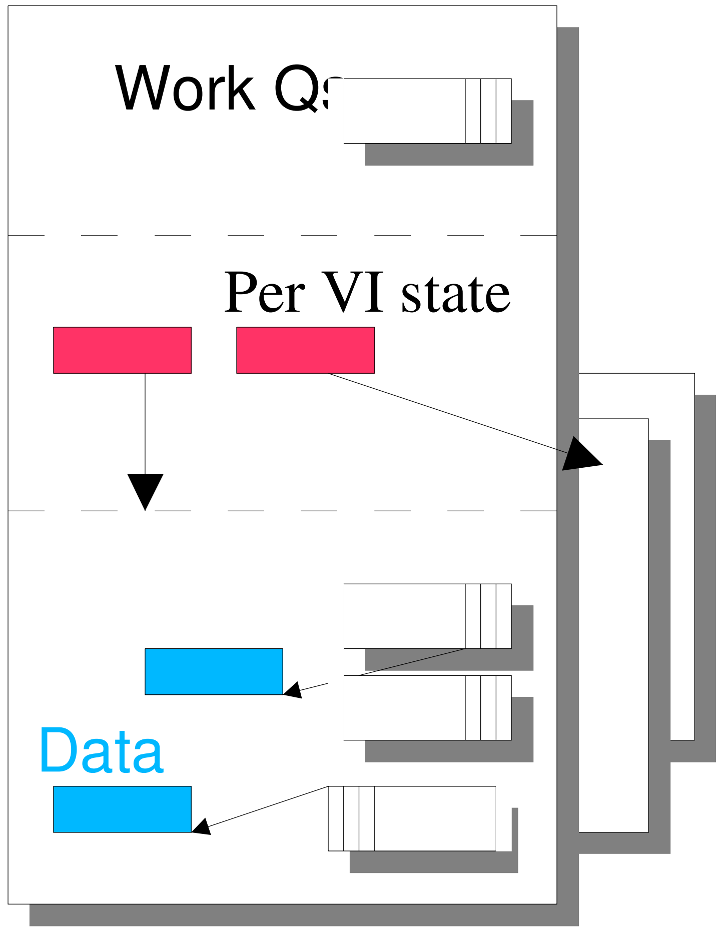


VIA– Big Picture

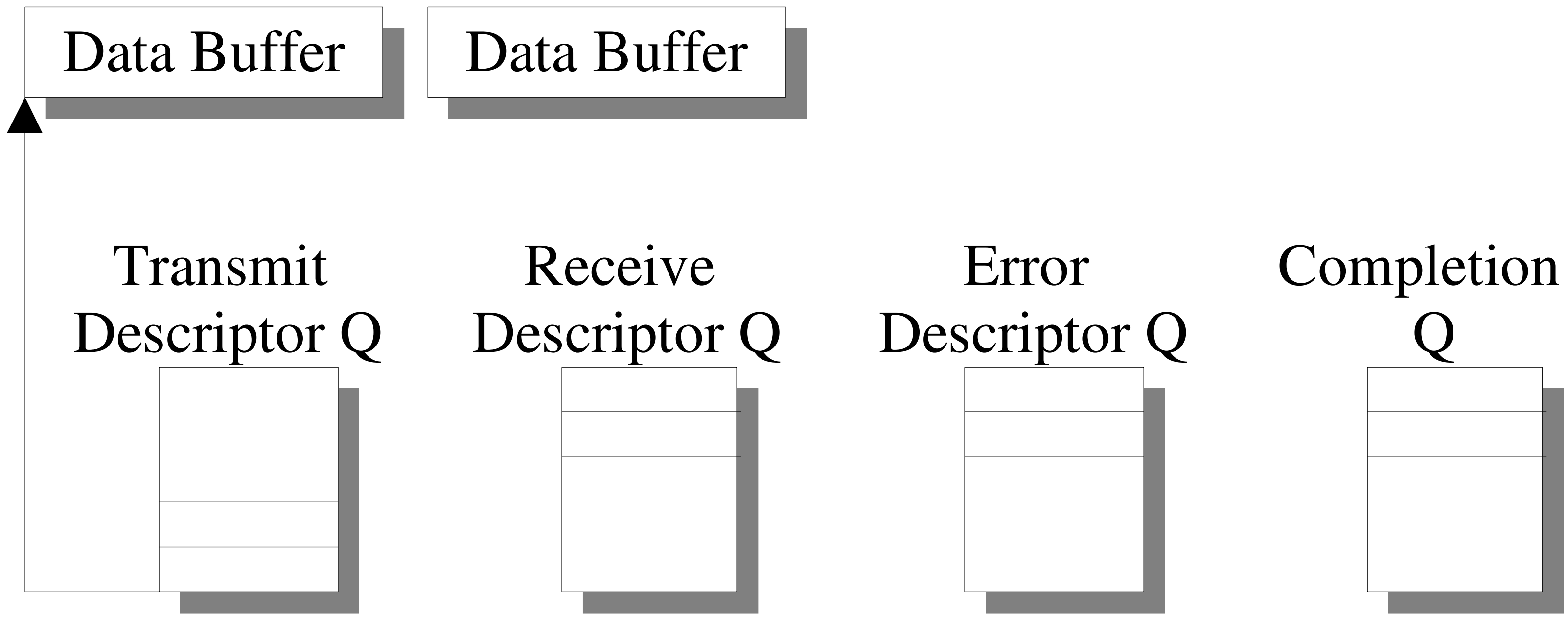
Device Space
(card memory)

Kernel Space

VIA registered region
of user memory



VIA Data Structures



NI and NI Memory

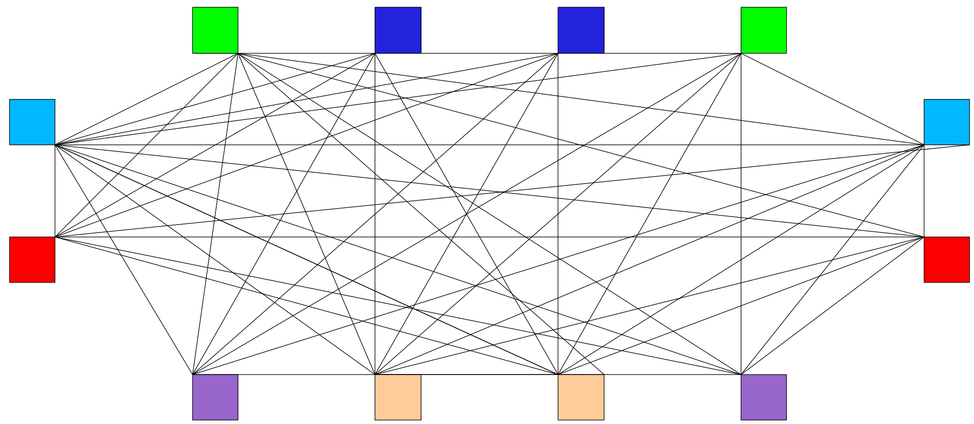
What are doorbells?

- Problem: how to notify system a descriptor is ready?
  - Status bits in the data queues themselves?
    - Expensive NI–host memory DMA?
  - "doorbell" is just an abstract notification mechanism
  - Servers same purpose as hardware FIFOs
    - More complex & costly?
    - VIA spec did not want to specify mechanism

VIA Completion Q

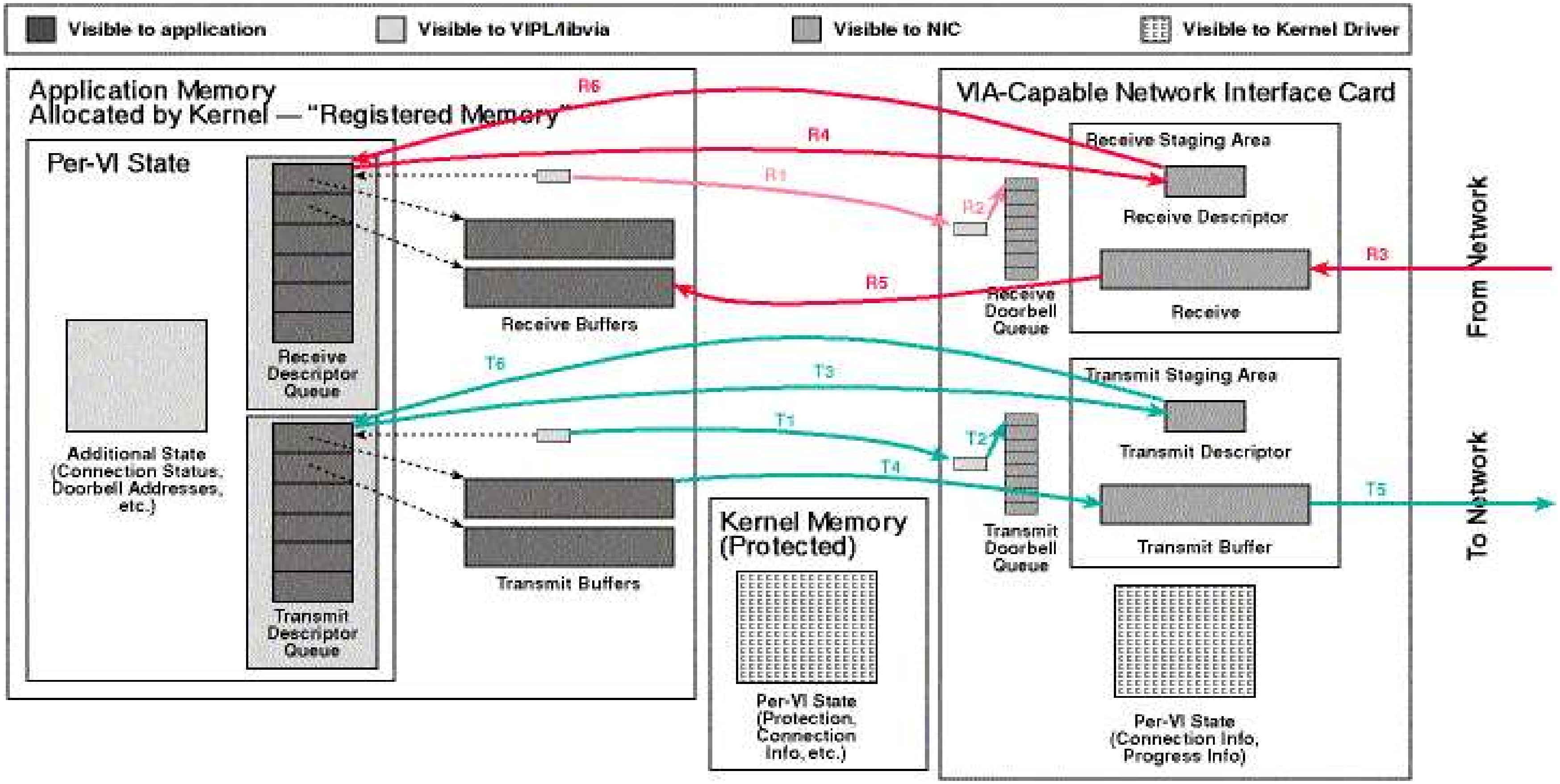
- Single Q for all completion events
  - Both send an receives complete on same Q
  - Designed to reduce overhead, one place to check everything.
- What about send vs. receive semantics?
  - Might actually increase effective overhead

N^2 communication



- How many point–to–point channels are required for an $O(N^2)$ abstraction?
  - VIA forgot this initially
  - Fixed with recent versions...

Berkeley VIA



Afterburner vs. VIA

- VIA abstraction has a lot of indirection
  - two layers of queueing
  - advantage is portability
- Afterburner abstraction much simpler
  - FIFOs are a simple door bell
  - Interaction with the memory system?
    - Protection
    - Size of on the on–card memory?
    - # of active connections, scalable comm?

Completion tradeoff

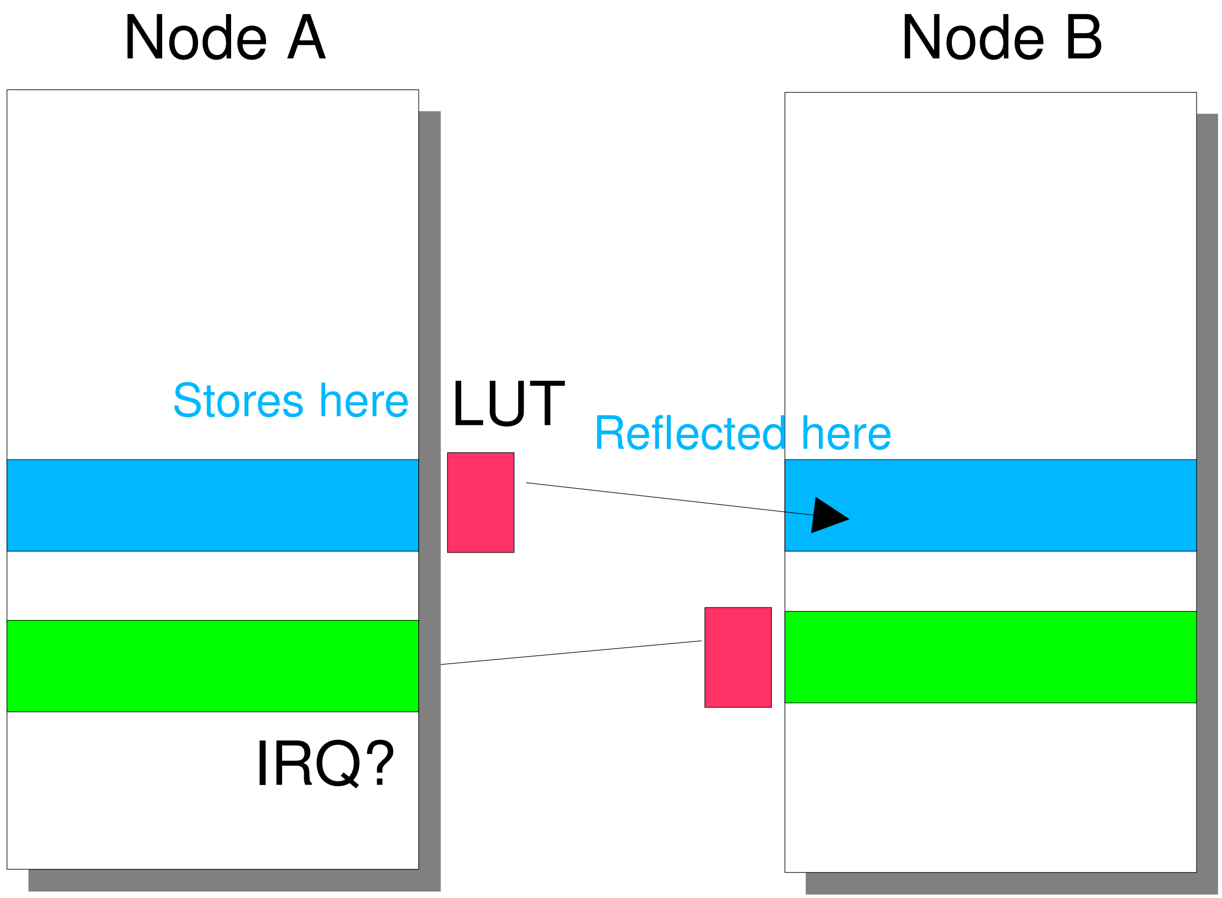
- Afterburner has two completion queues
  - Tx, RX require separate checks
  - Two loads to look for "done"
- Tradeoff:
  - Want to Transmit immediately
  - VIA must process RX completions in line with TX
  - Afterburner: TX can slip ahead

Shrimp

Kernel Space

User Space

No loads,
only stores



Reflective Memory

- Simple Idea
  - DEC memory channel, others
- Easy to implement
  - vs. load/store shared memory
- Semantic issues
  - How to build an N–to–1 Queues?
  - Notification?
  - Scalability?

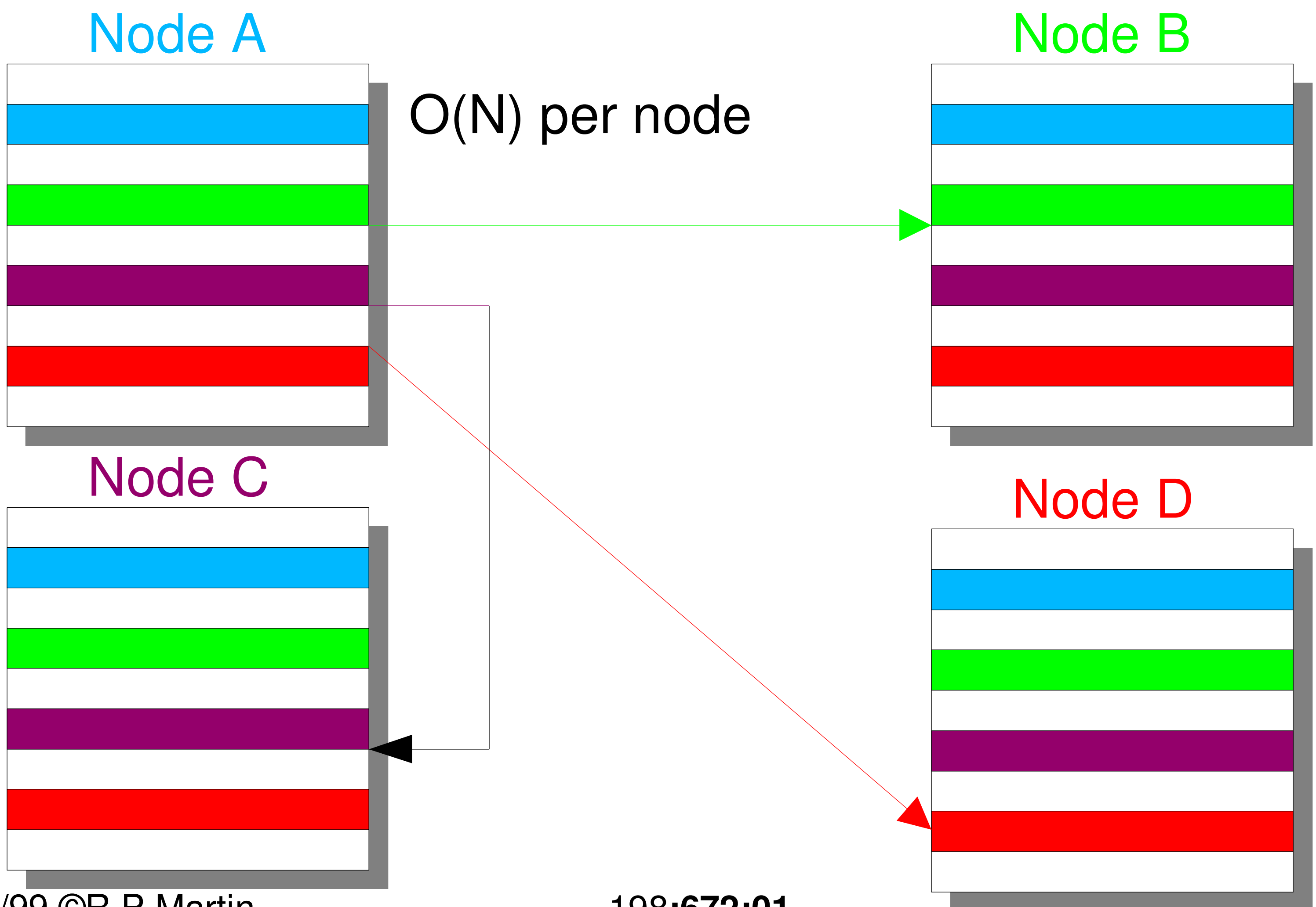
Performance Issues

- Trading off overhead and gap
  - In−memory data structures have low overhead
  - but... NI mem. access turn into a higher gap
- Better abstraction gets rid of descriptor Q's
  - Hardware FIFO on NI
  - Data in memory
  - I_2O hardware is like this ....
- Lock access in reflective memory?

Semantics?

- Need Messaging (e.g. MPI) on all systems.
- What if everyone sends to one?
  - Can't appear to drop.
- Communication is an $O(N^2)$ abstraction
  - How well do data structures scale?
    - Size $O(N)$ per node, or $O(1)$ per node?
  - How are o,g,L affected by scalability of DS?

Reflective Memory Scalability?



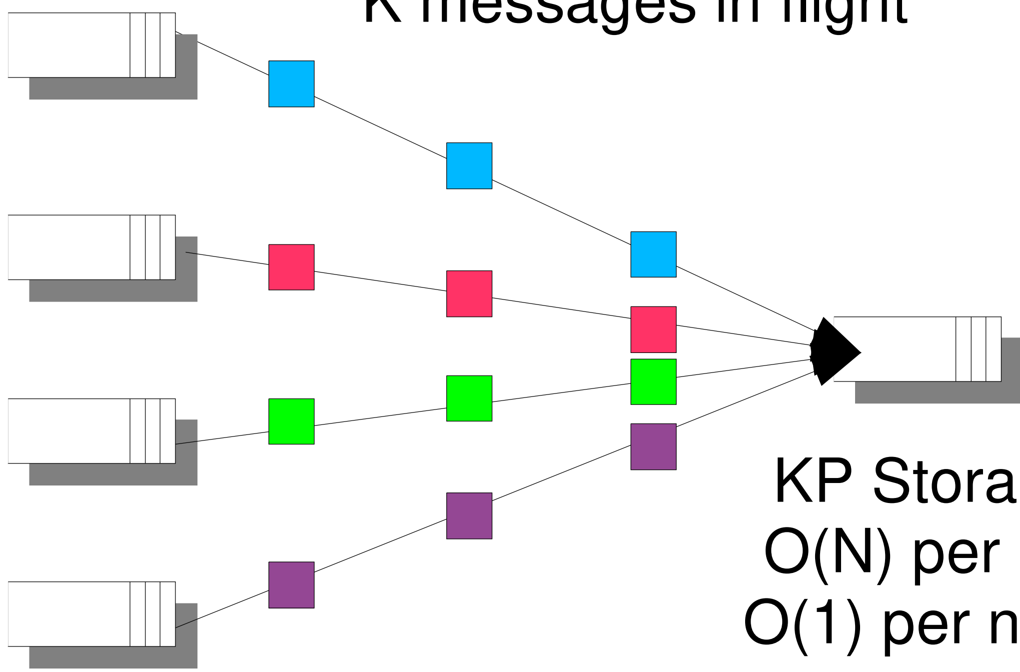
Memory Scalability

- Much worse: O(N) overhead.
  - Ok for 8, 16. 100? 500?
- Shared memory easier: Use locks
    - Acquire lock
    - Insert message into single Q
    - Release lock
- O(1) per Node
  - Lock contention?

KP Storage for Messaging

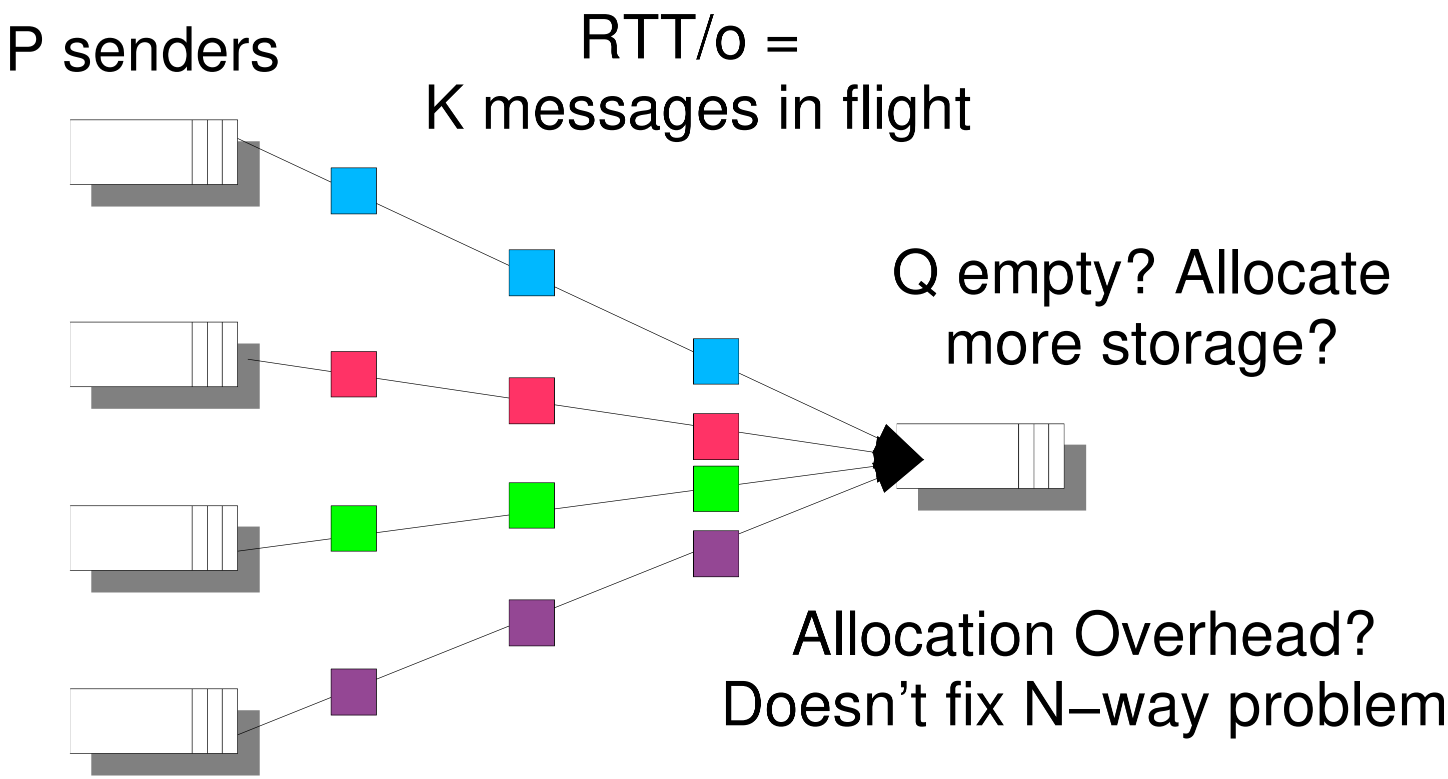
P senders

Pipeline requires RTT/o =
K messages in flight



KP Storage in input Q
O(N) per node storage
O(1) per node overhead

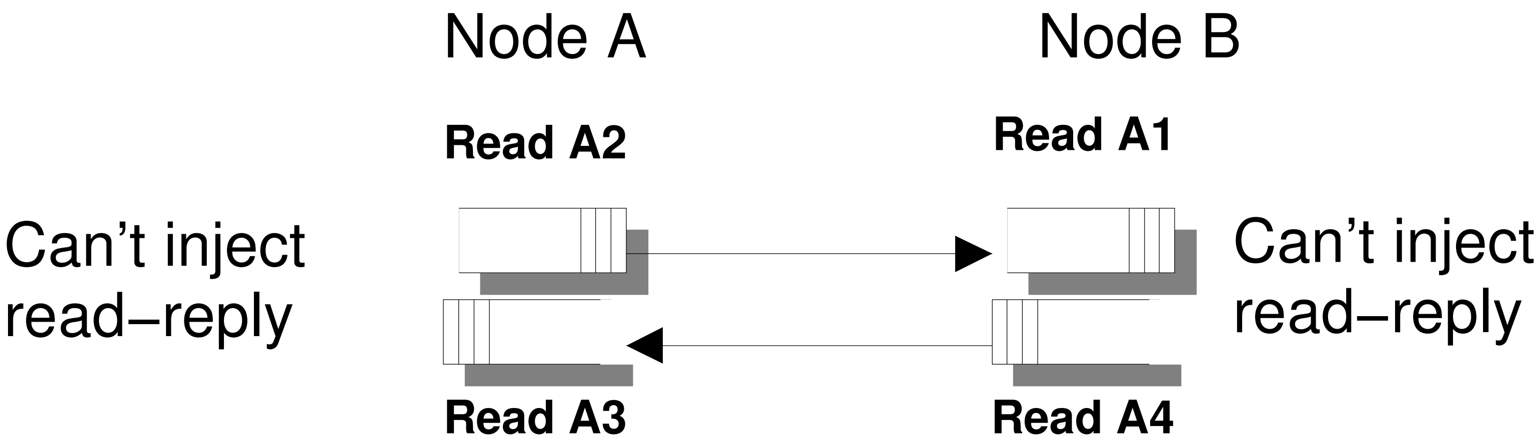
Try memory allocation?



Supporting N–way protocols

- N–way protocol is a circular dependency in the protocol of length N
  - Node A sends to B which generates a message to C which replies to A. Reply doesn't generate a message
- Shared memory protocols often 2–way
- Some file systems are 3–way
- Requires N independent communication paths
  - Deadlock or livelock possible if not careful.

Shared memory example



Deadlock?

–> Shared memory on TCP requires 2N sockets per node

Drop the message?

- Another solution
- Sender holds onto request message
- Receiver reserves O(1) NACK storage
- NACKs have priority
- Every incomming message:
  - deposits or
  - sends NACK.
- Retry performance?

3rd way: Connections

- In order to send data, must enable a connection
  - Used on the Meiko CS–2
  - Metro/NIFTY interface designs
- OK if done in hardware
  - But some problems in not enough channels/queuing
  - If busy, probably overloaded anyway.

Availability/Managebility

- Sparse in the world of research
- What is "available" communication?
  - Tx/Rx in the presence of hardware faults?
  - Software Failures?
- What is "manageable communication"
  - Automatic configuration?
  - Join/leave the group?

Next Week

- Storage sub–systems
  - Petal: A virtual block device
  - Storage Subsystems (PhD Quals talk)
  - Swarm (Client based File–System)

Kernel Message Structure

Linux: skbuff (2.2.X)

STREAMS: msg buffer