Networking for Physics Programmers

Glenn Fiedler
Sony Santa Monica
www.gafferongames.com

Tuesday, March 16, 2010

This talk is about how to network a physics simulation over the internet.

This talk is in three parts:

1) First I will show you how the Internet works - How much bandwidth you can use, what network conditions are like and whether to use TCP or UDP for sending packets etc...

2) Next I will show you how to synchronize a physics simulation running on one computer such that it can be viewed on another.

3) Finally, I'll describe one possible networking model (out of many) in reasonable detail. This networking model is well suited to P2P cooperative games with lots of physics object interactions. I'll show you how this network model avoids O(n) simulation cost per-player, how it hides latency when players interact with physically simulation objects, and how it supports late joins - all without requiring a dedicated server.

Lets begin!

DEMO

Tuesday, March 16, 2010

This demo shows what we are trying to network!

It is designed very specifically to show a case where players interact at close range with physically simulated objects using the open source library ODE (Open Dynamics Engine)

The objects can roll, tumble and stack. Their motion is highly non-linear. Traditional networking techniques break down for this sort of motion.

There are one million cubes in this demo. But we can only afford to simulate and render a few at any time, so only objects within the activation circle are simulated and rendered.

Objects outside the circle are deactivated. The entire world is persistent so if I move some objects over here, then move away -- when I return to that area the objects are in the same place I left them. (try doing this by holding SPACE + arrow keys in the demo to clear a line... then return along that line to inspect the cubes are as they were left)

This persistence is quite important for the rest of this talk. Please note that objects are not “created” or “destroyed” as they fade-in and out - they always exist. They simply deactivate when they are outside your circle.

There is an array of one million cubes in this demo in memory. Each cube has a unique id corresponding to its index in this array. While an object is active it is assigned an “active id” which is the index of the active cube in the pool of active objects. This active id is transient and will be different each time an object activates.

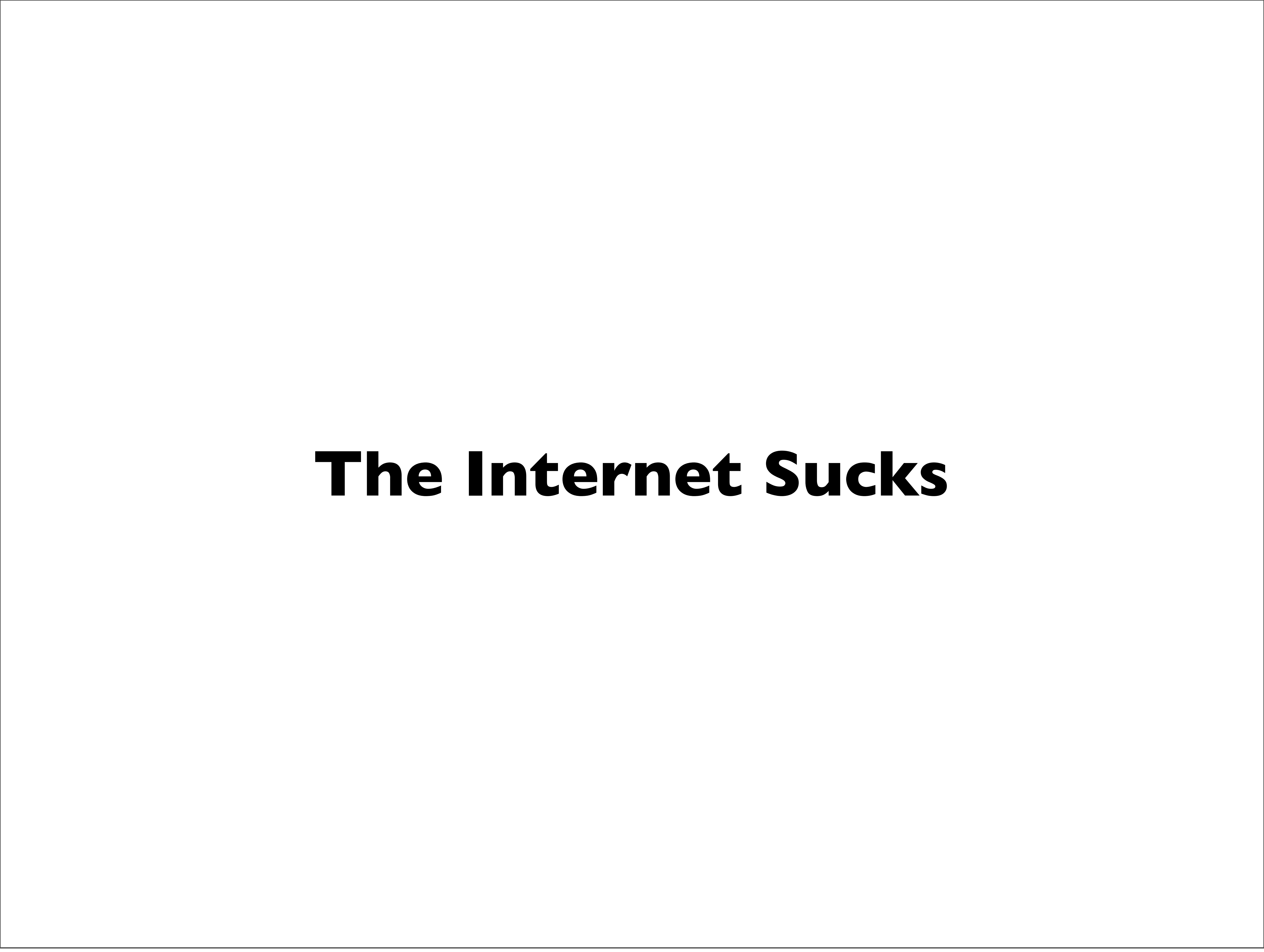
In effect this demo shows a very basic open world game engine. So consider therefore that this talk is not only going to show you how to network a physics simulation, but how to do so in an open world game such that you may synchronize one million cubes and support late join, without requiring existing players in the game to pause or to zip up and send the state for up to one million dirty cubes to the client when they join.

Everything about this networking model is designed to “stream in” the minimal amount of changes as required as players move around the world.

You can download the demo and source code from www.gafferongames.com

The demo is open source with a creative commons non-commercial use clause.

Please contact me on glenn.fiedler@gmail.com if you have any questions about the demo, I’m happy to answer your questions.



The Internet Sucks

Tuesday, March 16, 2010

PART ONE

Before we begin networking, I have some bad news for you...

The internet sucks.

I will prove it to you!

Sony Bandwidth Probe

30th Jan - 18th Feb 2010

2.7 Million Samples

First I will show you exactly how much bandwidth is available.

It is less than you think!

I have some bandwidth statistics collected from four Sony titles: **Pain, Warhawk, Fat Princess and GT5 Prolog**

Upload and bandwidth samples were collected over a period of 20 days: 01/30/2010 and 02/18/2010

There are over 2.7 million samples in total across all four Sony territories:

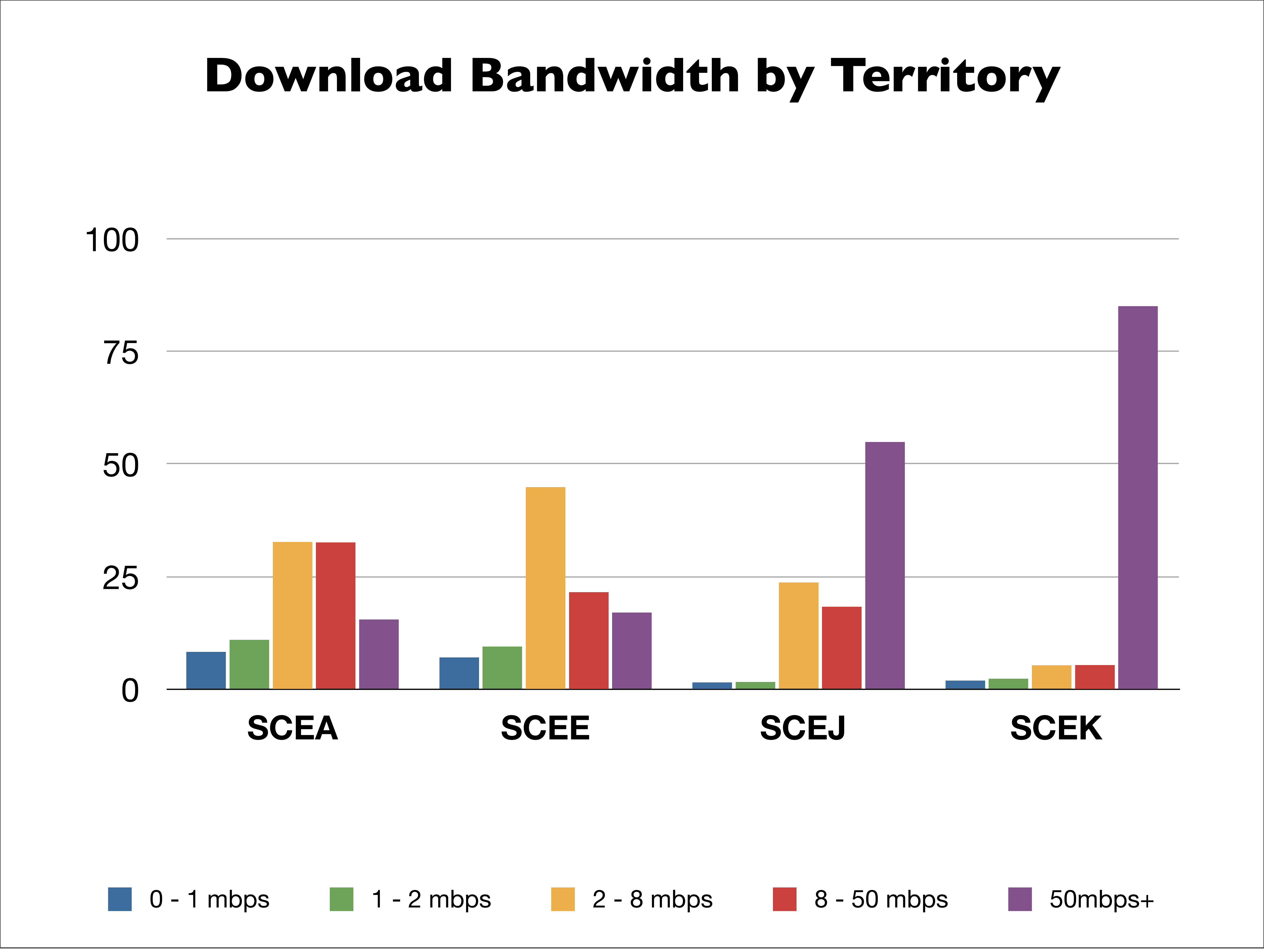
SCEA (USA)
1,022,240 upload samples
339,207 download samples

SCEE (Europe)
671,969 upload samples
310,240 download samples

SCEJ (Japan)
284,973 upload samples
29,076 download samples

SCEK (Korea)
36,186 upload samples
18,808 download samples

TOTAL SAMPLES: 2,710,699



Download Bandwidth by Territory

Download bandwidth probe samples.

When the title is run it connects to a nearby Sony server in the local region and performs a download bandwidth test.

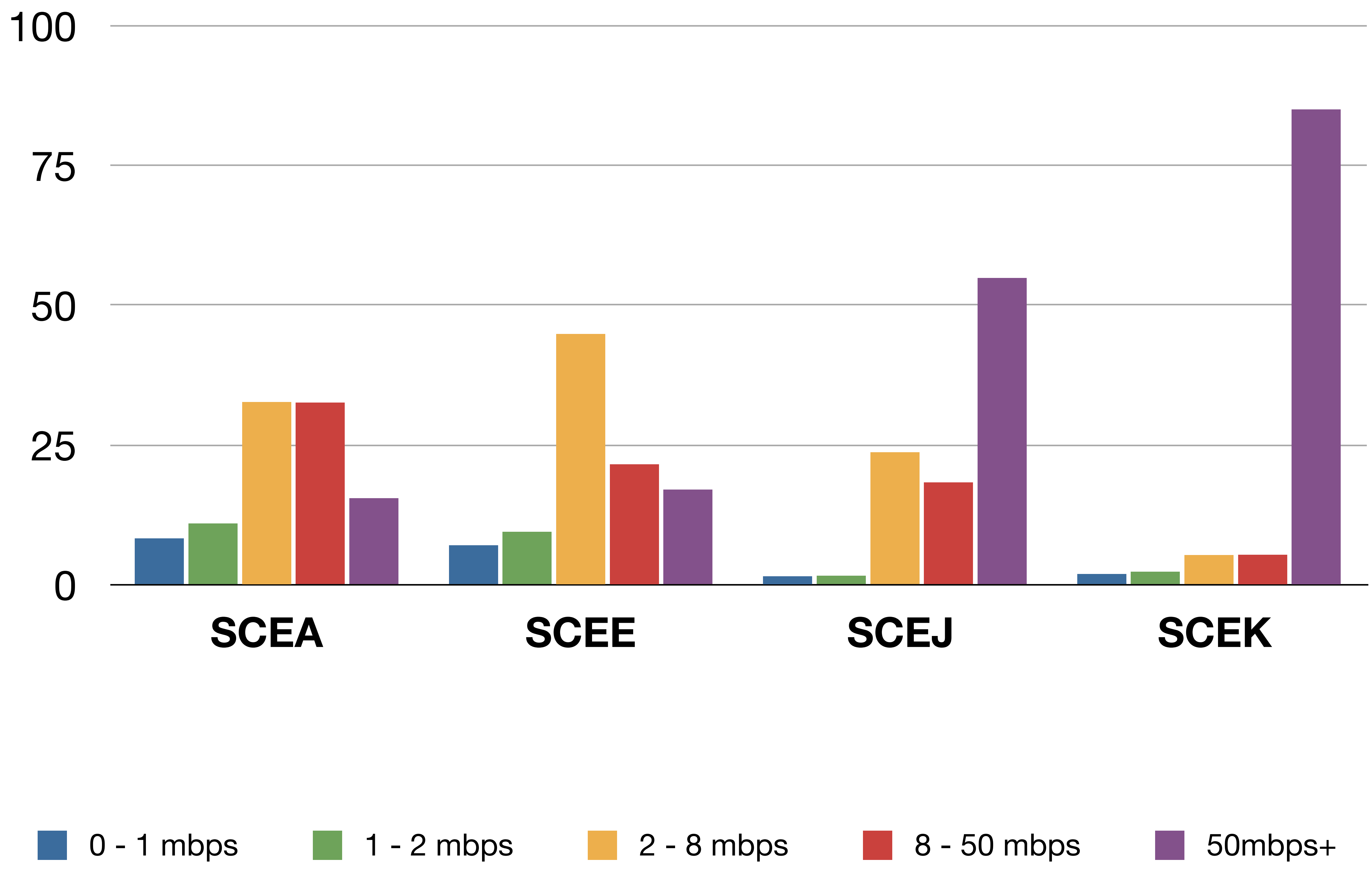
The tested bandwidth represents not the “advertised” speed of the ISP connection, but the actual bandwidth as verified between the PS3 running the title and the bandwidth probe server in the same region.

The scale from left to right is:

0-1 mbps
1-2 mbps
2-8 mbps
8-50 mbps
50mbps +

Note: I’m being particularly loose with kbps and mbps -- you get the general idea though.

Download Bandwidth by Territory



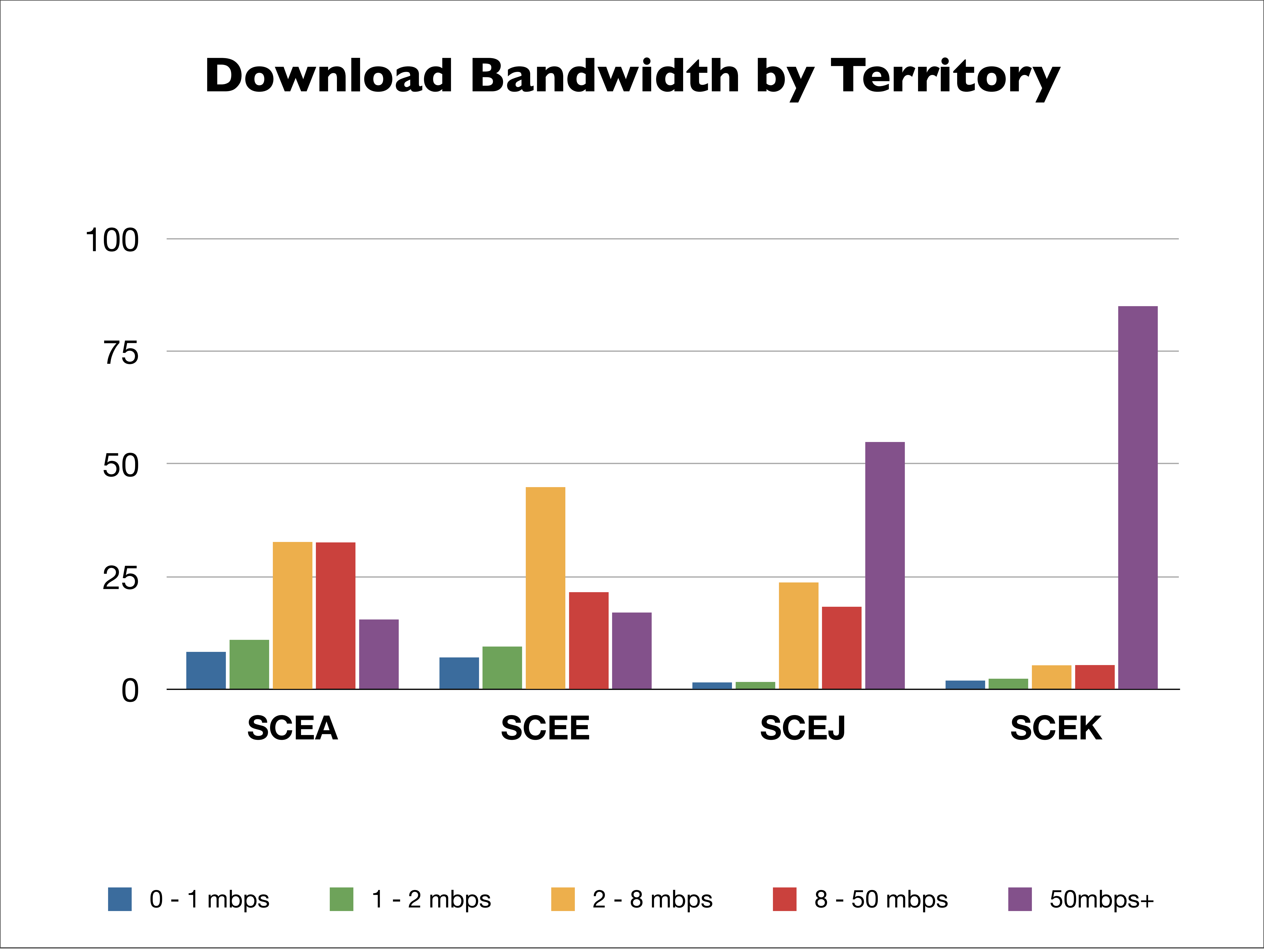
Tuesday, March 16, 2010

Here are the download bandwidth stats for the USA:

[SCEA Download]

0 to 256 kbps: 3244
257 to 512 kbps: 5224
513 to 768 kbps: 9677
769 to 1024 kbps: 9891
1025 to 1536 kbps: 24548
1537 to 2048 kbps: 12613
2049 to 8192 kbps: 110960
8193 to 49152 kbps: 110519
49153 kbps and higher: 52531
total: **339207**

99% have **256kbps** download or greater
95.5% have **512kbps** download or greater
92% have **1mbps** download or greater
15.5% have **50mbps** download or greater



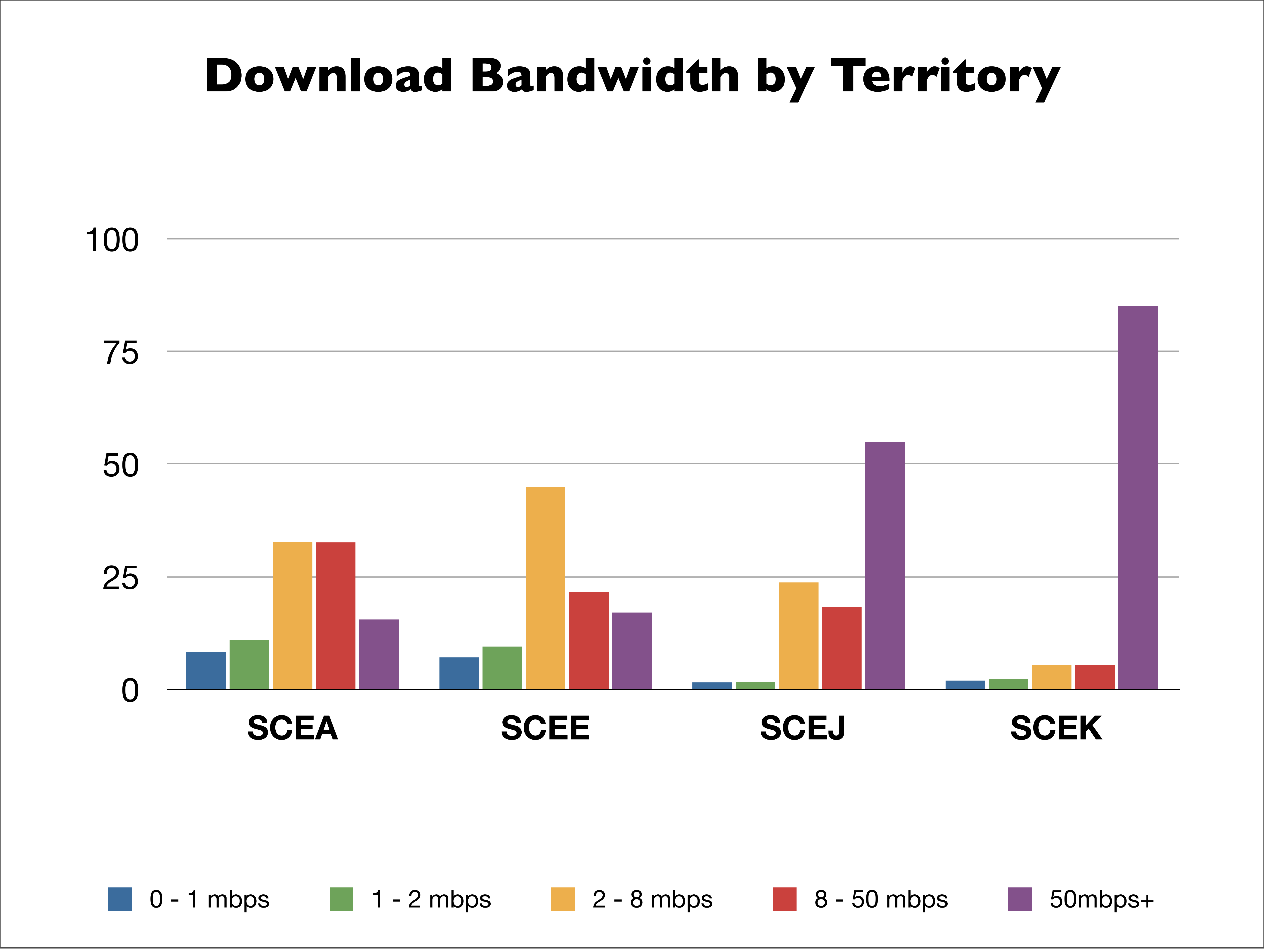
Download Bandwidth by Territory

Europe

[SCEE Download]

0 to 256 kbps: 2612
257 to 512 kbps: 4842
513 to 768 kbps: 5488
769 to 1024 kbps: 8937
1025 to 1536 kbps: 12357
1537 to 2048 kbps: 17043
2049 to 8192 kbps: 139246
8193 to 49152 kbps: 66903
49153 kbps and higher: 52812
total: **310240**

99.15% have **256kbps** download or greater
97.6% have **512kbps** download or greater
92.95% have **1mbps** download or greater
17% have **50mbps** download or greater



Download Bandwidth by Territory

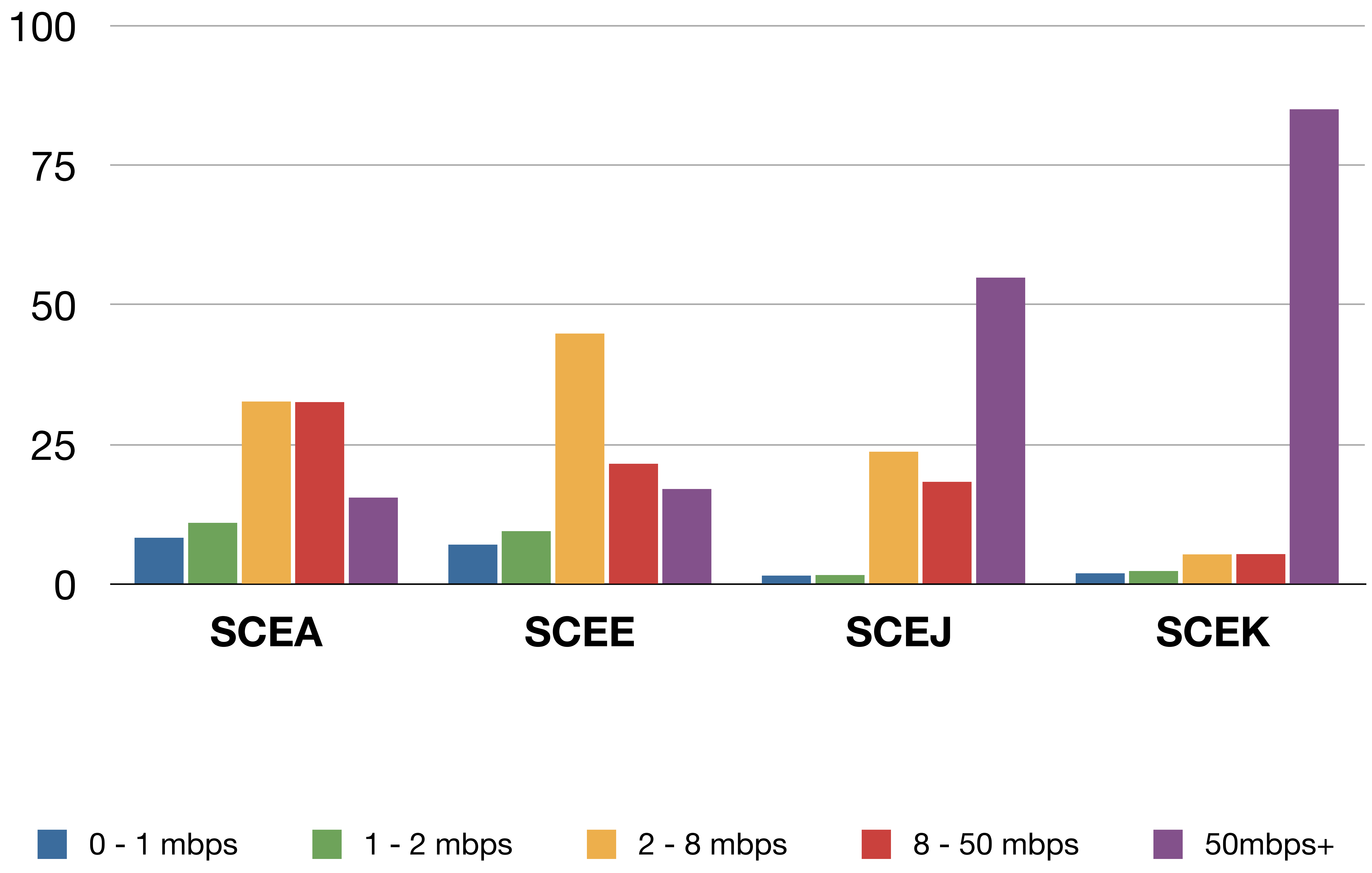
Japan

[SCEJ Download]

0 to 256 kbps: 59
257 to 512 kbps: 140
513 to 768 kbps: 70
769 to 1024 kbps: 176
1025 to 1536 kbps: 235
1537 to 2048 kbps: 229
2049 to 8192 kbps: 6895
8193 to 49152 kbps: 5318
49153 kbps and higher: 15954
total: **29076**

96.87% have **2mbps** or greater download
73.1% have **8mbps** or greater download
54.87% have **50mbps** or greater download

Download Bandwidth by Territory



Tuesday, March 16, 2010

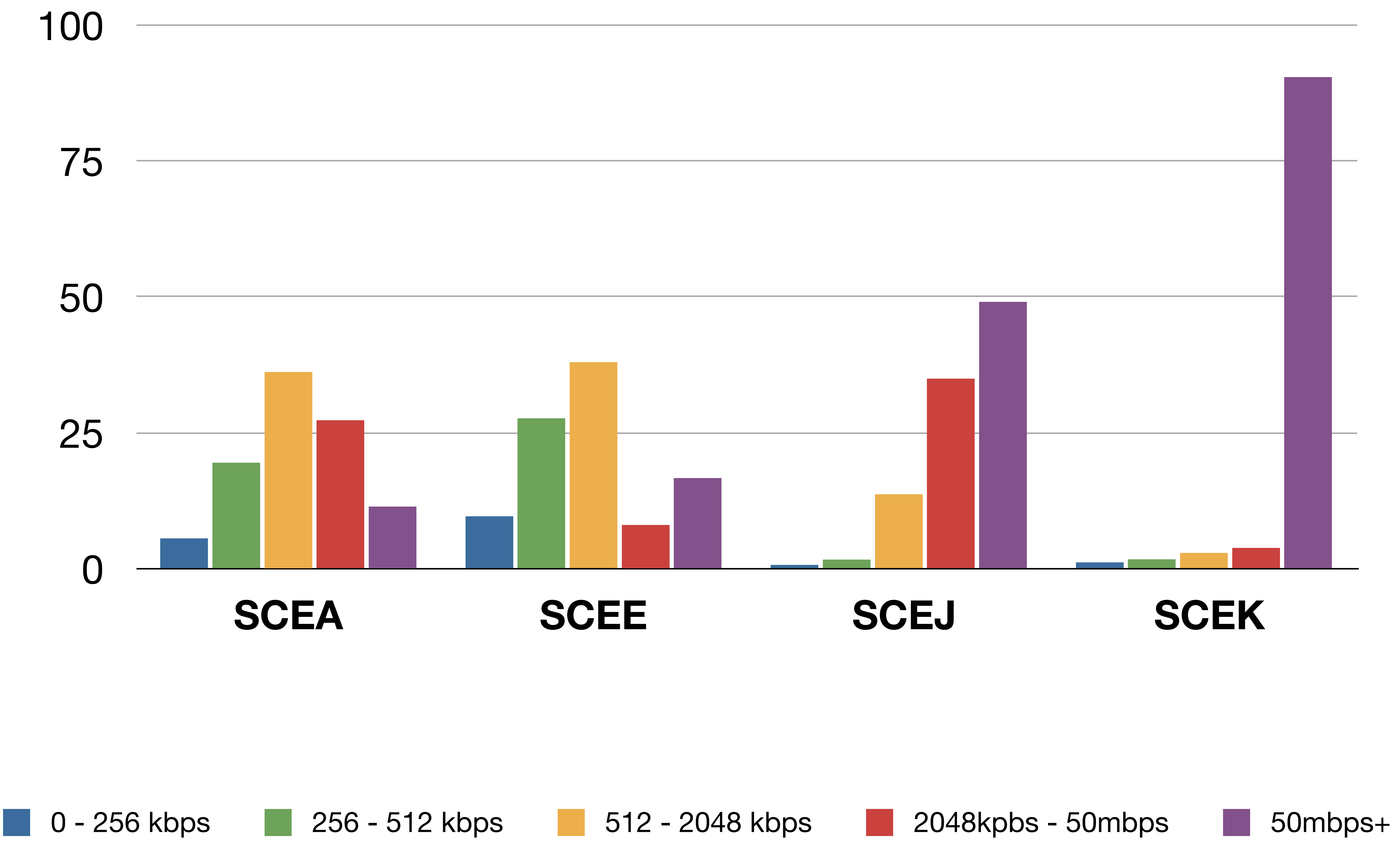
Korea

[SCEK Download]

0 to 256 kbps: 123
257 to 512 kbps: 34
513 to 768 kbps: 42
769 to 1024 kbps: 162
1025 to 1536 kbps: 121
1537 to 2048 kbps: 319
2049 to 8192 kbps: 999
8193 to 49152 kbps: 1011
49153 kbps and higher: 15997
total: **18808**

85% have **50mbps** or greater upload (!!)

Upload Bandwidth by Territory



Tuesday, March 16, 2010

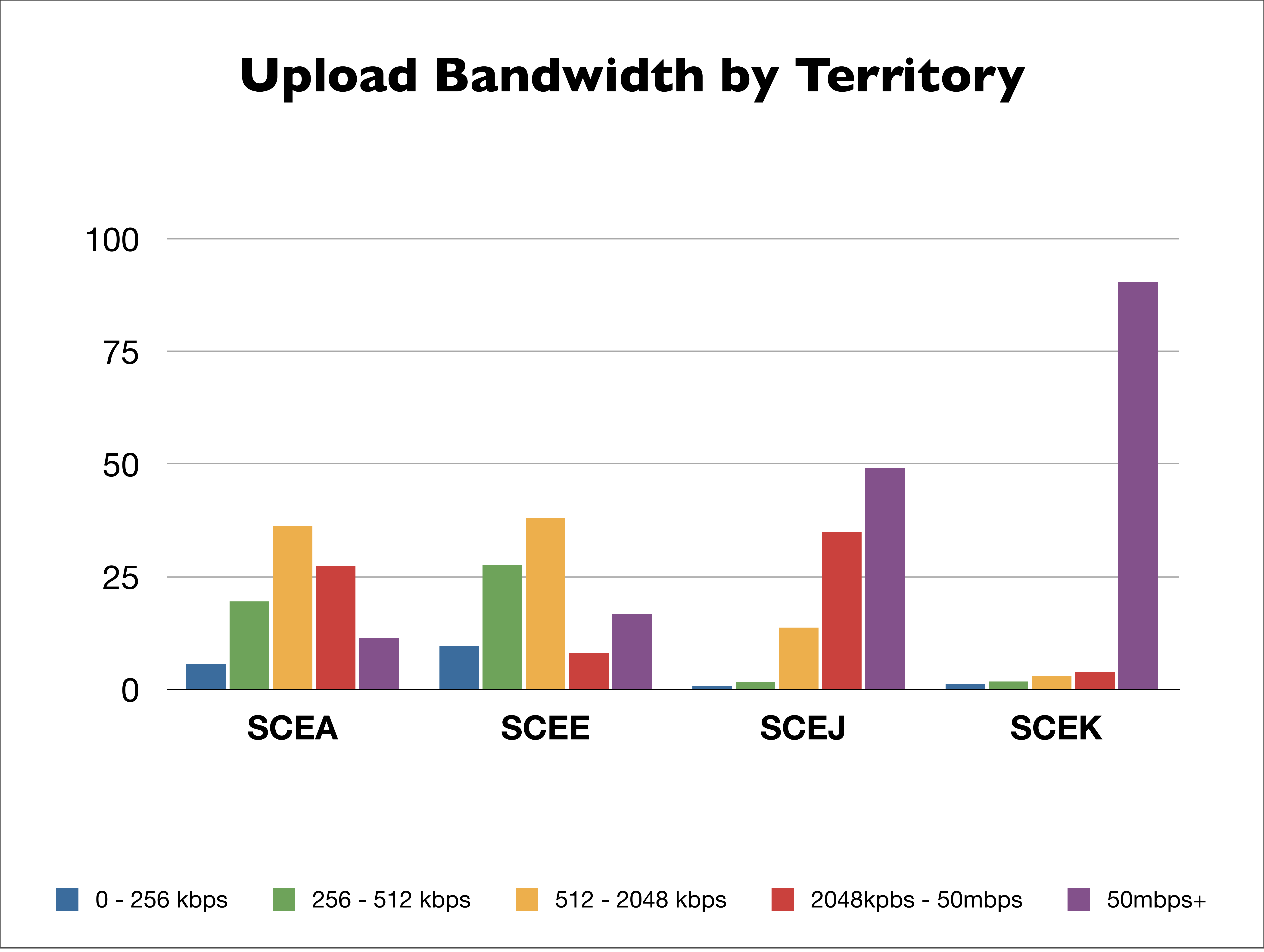
Upload bandwidth samples now...

IMPORTANT: The scale is now different!

0 – 256 kbps
256 – 512 kbps
512 – 2048 kbps
2048 – 50mbps
50 mbps +

Why?

Less upload than download bandwidth, in general.



Upload Bandwidth by Territory

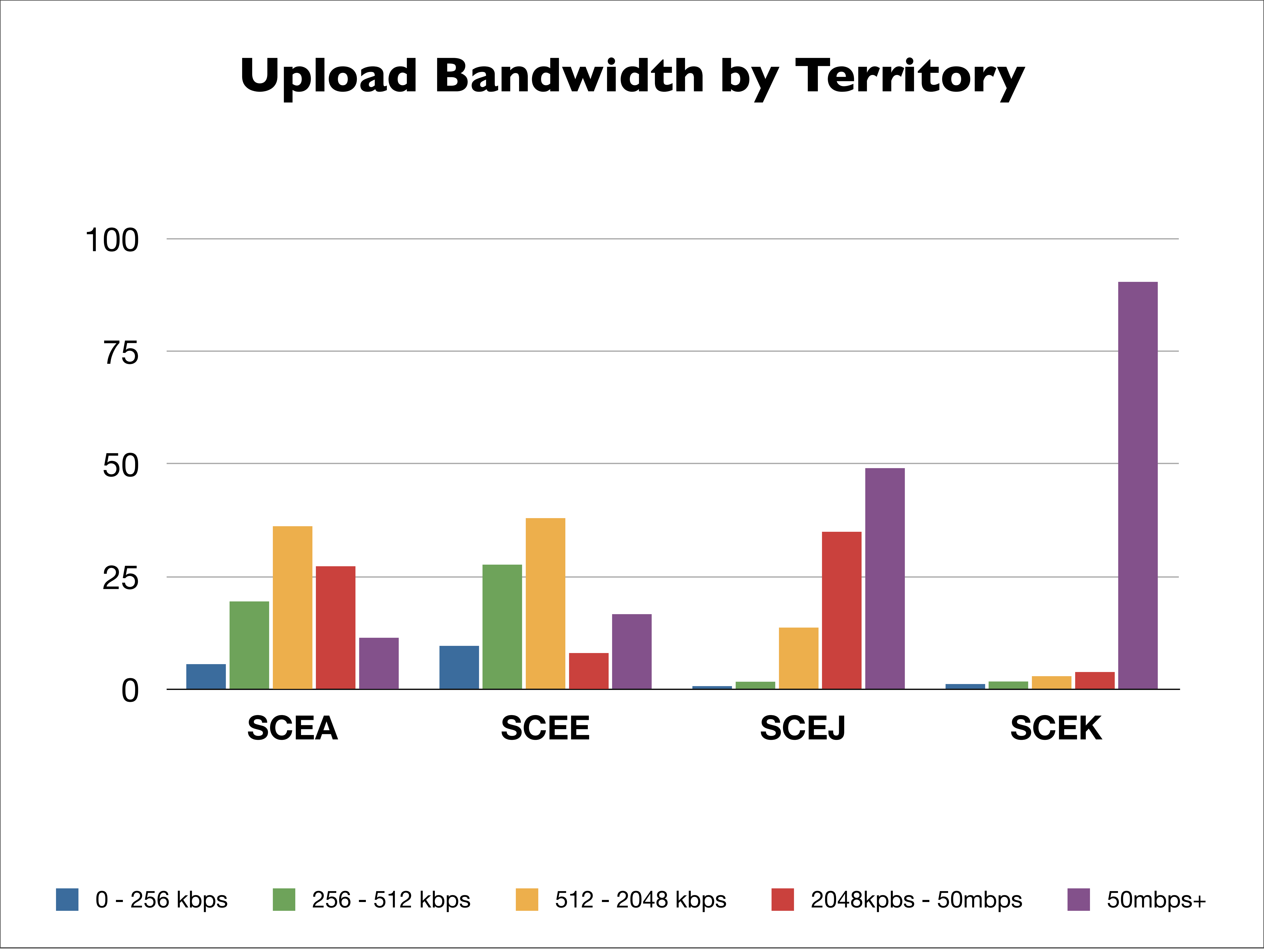
Tuesday, March 16, 2010

USA

[SCEA Upload]

0 to 64 kbps: 11382
65 to 128 kbps: 15875
129 to 192 kbps: 13057
193 to 256 kbps: 16471
257 to 320 kbps: 25956
321 to 384 kbps: 67586
385 to 448 kbps: 68718
449 to 512 kbps: 36776
513 to 768 kbps: 162619
769 to 1024 kbps: 86700
1025 to 1536 kbps: 74117
1537 to 2048 kbps: 46740
2049 to 8192 kbps: 207959
8193 to 49152 kbps: 71228
49153 kbps and higher: 117055
total: **1022240**

99% of samples have **64kbps** upload or greater
97.3% of samples have **128kbps** upload or greater
94.4% of samples have **256kbps** upload or greater
75% have **512kbps** upload or greater
11% have **50mbps** or greater upload



Upload Bandwidth by Territory

Tuesday, March 16, 2010

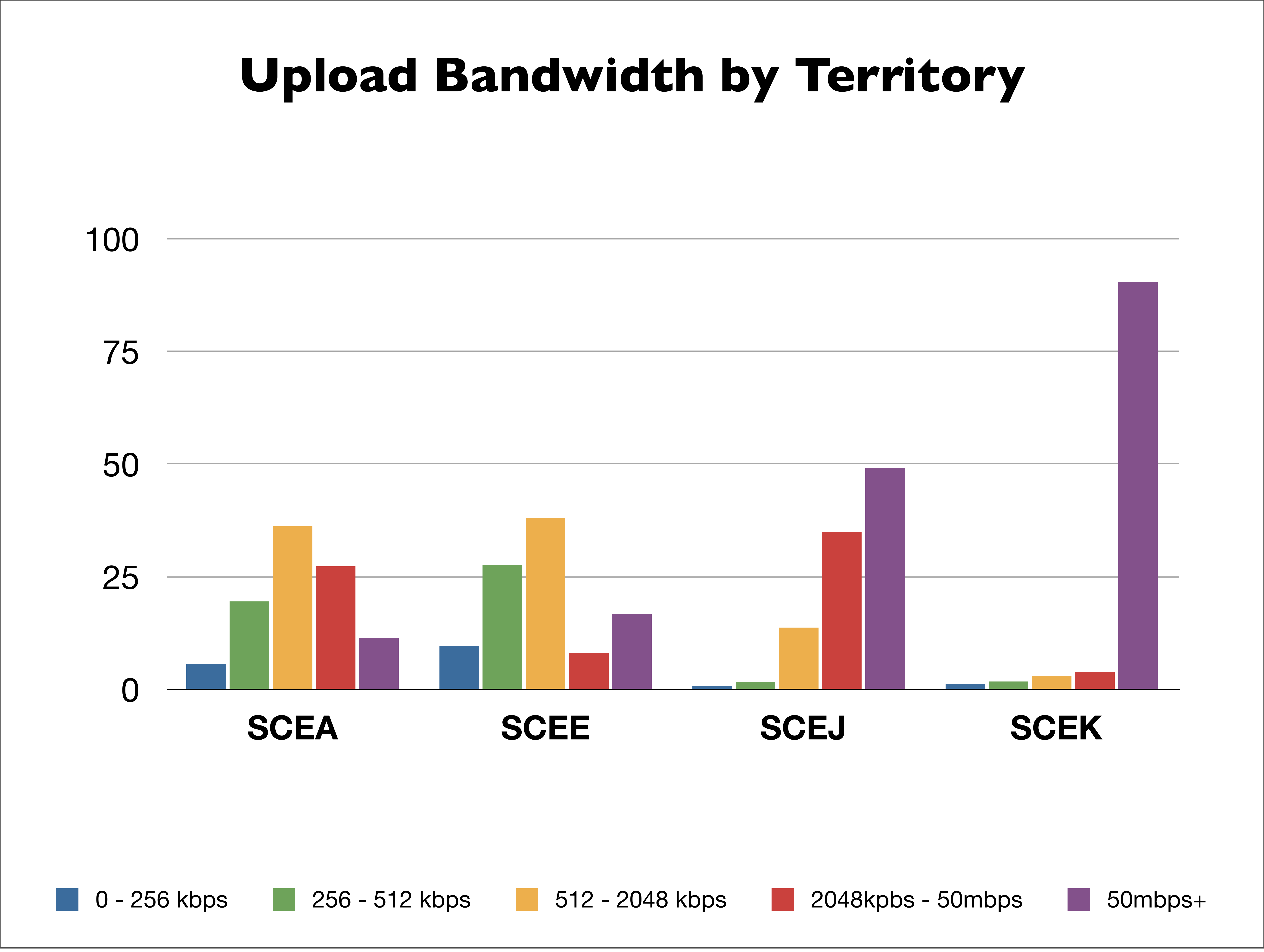
Europe

-------------------------

[SCEE Upload]

0 to 64 kbps: 4331
65 to 128 kbps: 14360
129 to 192 kbps: 21310
193 to 256 kbps: 24708
257 to 320 kbps: 35718
321 to 384 kbps: 50931
385 to 448 kbps: 57975
449 to 512 kbps: 41187
513 to 768 kbps: 101331
769 to 1024 kbps: 113460
1025 to 1536 kbps: 26815
1537 to 2048 kbps: 13687
2049 to 8192 kbps: 27415
8193 to 49152 kbps: 26738
49153 kbps and higher: 112003
total: **671969**

99.4% have **64kbps** upload or greater
97.2% have **128kbps** upload or greater
89% have **256kbps** upload or greater
62.72% have **512kbps** upload or greater
16.7% have **50mbps** upload or greater



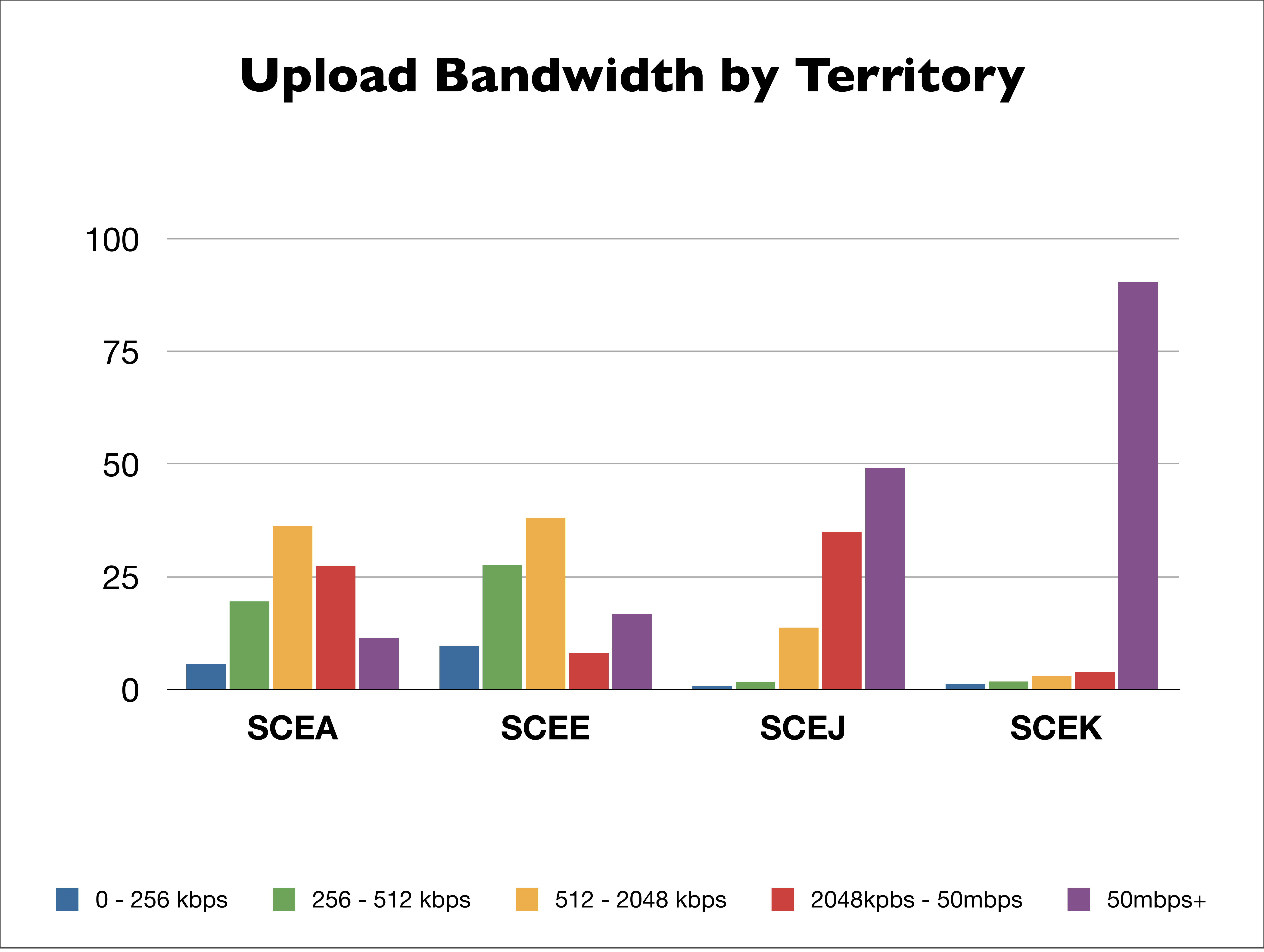
Upload Bandwidth by Territory

Japan

[SCEJ Upload]

0 to 64 kbps: 262
65 to 128 kbps: 241
129 to 192 kbps: 540
193 to 256 kbps: 865
257 to 320 kbps: 553
321 to 384 kbps: 675
385 to 448 kbps: 1741
449 to 512 kbps: 1706
513 to 768 kbps: 10378
769 to 1024 kbps: 17449
1025 to 1536 kbps: 5142
1537 to 2048 kbps: 6006
2049 to 8192 kbps: 14300
8193 to 49152 kbps: 85247
49153 kbps and higher: 139868
total: **284973**

99.9% have **64kbps** upload or greater
97.7% have **512kbps** upload or greater
84% have **2048kbps** upload or greater
49% have **50mbps** upload or greater



Upload Bandwidth by Territory

Tuesday, March 16, 2010

Korea

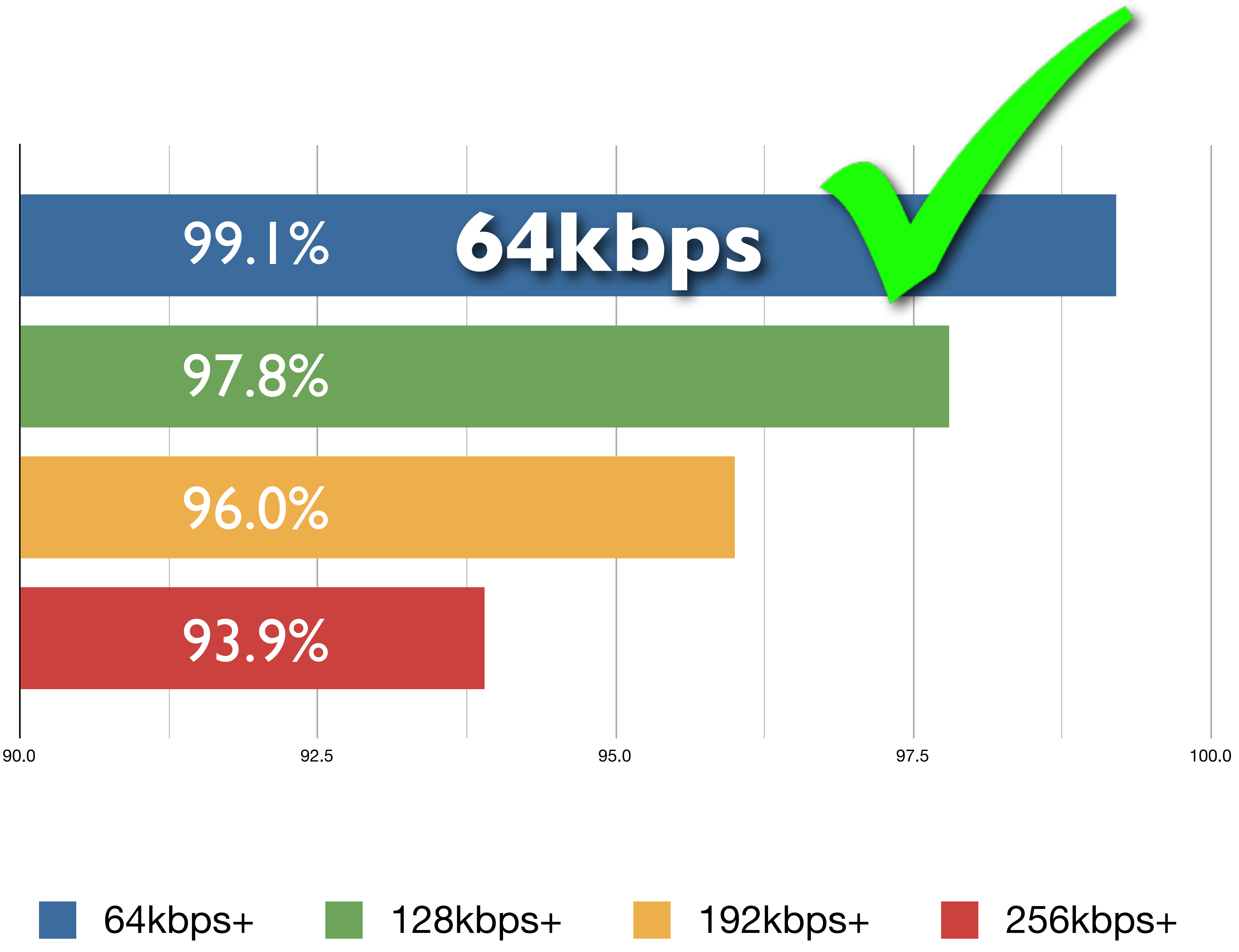
[SCEK Upload]

0 to 64 kbps: 13
65 to 128 kbps: 58
129 to 192 kbps: 37
193 to 256 kbps: 308
257 to 320 kbps: 89
321 to 384 kbps: 144
385 to 448 kbps: 265
449 to 512 kbps: 119
513 to 768 kbps: 381
769 to 1024 kbps: 289
1025 to 1536 kbps: 197
1537 to 2048 kbps: 186
2049 to 8192 kbps: 388
8193 to 49152 kbps: 989
49153 kbps and higher: 32723
total: **36186**

90.4% have **50mbps** or greater upload (*see note!)

NOTE: in previous download samples for SCEK we found that 85% have 50mpbs or greater DL
but now we have 90% 50mbps upload or greater. this does not make sense!
we believe this to be an anomaly caused by not enough DL samples vs. UL in SCEK
we conclude that is most likely that in SCEK 90% have >= 50mbps up *AND* down! **GO KOREA!!!**

Minimum Spec Up/Down (%)



Tuesday, March 16, 2010

99.1% can support 64kbps up and down.

This is my recommended minimum spec. Aim for this!

That's roughly 256 byte packets @ 30 packets per-second*

IMPORTANT: there are significantly more samples are USA and Europe than Japan and Korea
Therefore this combined % is heavily biased towards the Western market. But then again,
the situation is so good in Japan and Korea - the Western market is the one you have to worry about! :)

*Does not include packet header, which typically takes up around ~40 bytes per-packet on PSN, it's more like 220 bytes or so

[Upload Samples]

0 to 64 kbps: 11382 + 4331 + 262 + 13 = 15988
65 to 128 kbps: 15875 + 14360 + 241 + 58 = 30534
129 to 192 kbps: 13057 + 21310 + 540 + 37 = 34944
193 to 256 kbps: 16471 + 24708 + 865 + 308 = 42352
total samples: 1022240 + 671969 + 284973 + 36186 = 2015368

99.2% - **64** kbps or greater upload
97.8% - **128** kbps or greater upload
96.0% - **192** kbps or greater upload
93.9% - **256** kbps or greater upload

[Download Samples]

0 to 256 kbps: 3244 + 2612 + 59 + 123 = 6038
total samples: 339207 + 310240 + 29076 + 18808 = 697331

99.1% - **256** kbps or greater download

UDP Reflector Experiment

Coded a simple UDP Reflector
VPS in Seattle, Atlanta and Sydney
Sent 256 byte packets @ 30pps from LA
and recorded when they bounced back...
(This is roughly 64kbps)

Tuesday, March 16, 2010

I coded a simple UDP reflector which bounced packets back to me

Then I setup VPS servers for about $10 a month in Seattle, Atlanta, and Sydney which ran my reflector code.

I sent a 64kbps stream of 256 bytes @ 30pps from my computer in Los Angeles over my WiFi router + time warner cable

The packets bounced off the reflectors and back to me and I recorded the round trip times for each packet

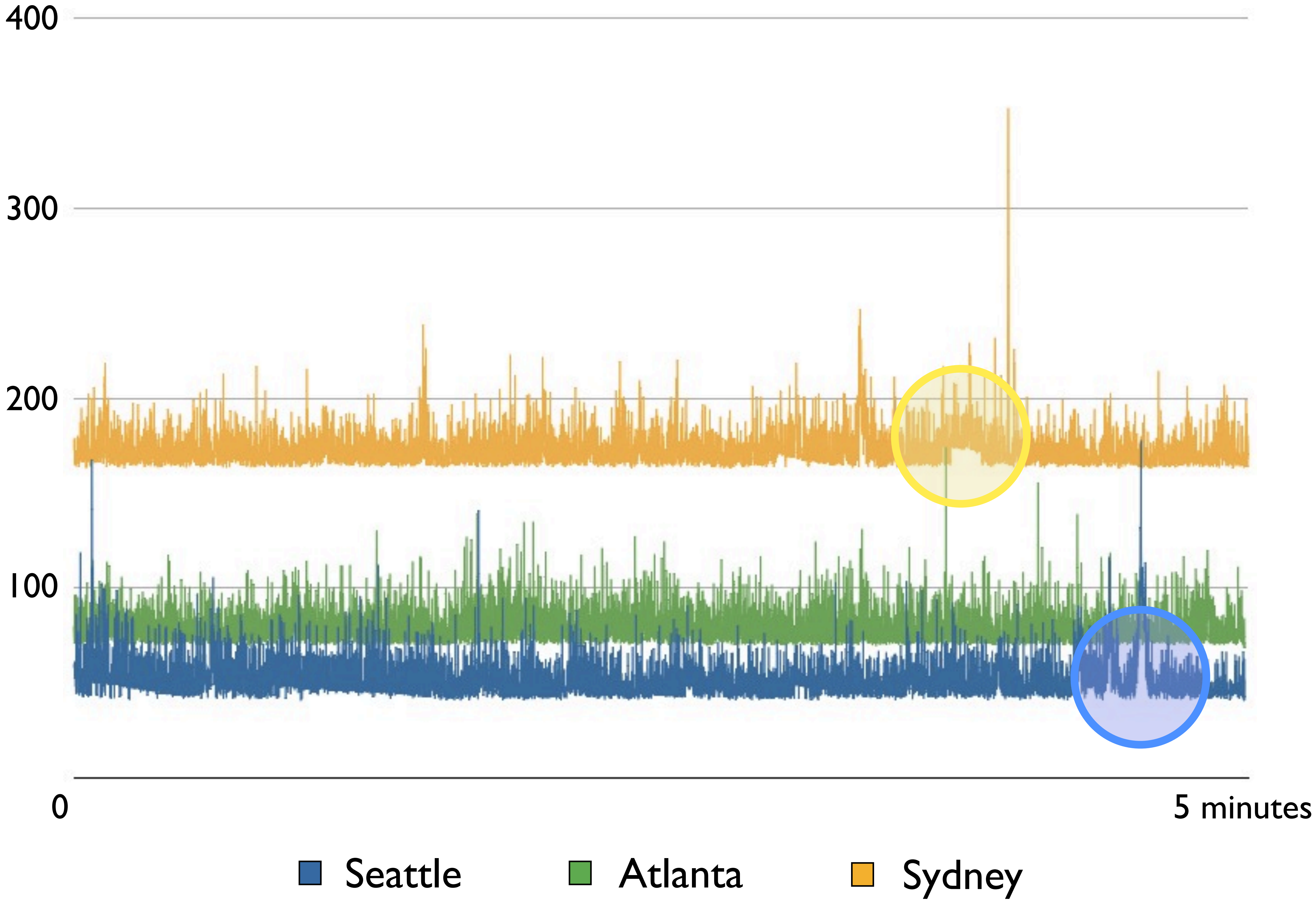
The Internet Sucks

Tuesday, March 16, 2010

Unfortunately what I discovered was not pretty!

The internet sucks, and not just because bandwidth is limited...

Round Trip Time - WiFi (ms)



Tuesday, March 16, 2010

Here is what I got from my home connection. A fast Time Warner cable 10mbit connection (!)

The graph shows the time it takes for a packet to travel to the reflector and back

This is the round trip time or “RTT”

I wrote a Ruby script to analyze the samples and and export the CSV which created this graph

Here are the results for this run:

5 minutes @ 30pps = 9000 samples

Mean RTT:

52ms to Seattle
79ms to Atlanta
173ms to Sydney

Packet loss not really a problem at all:

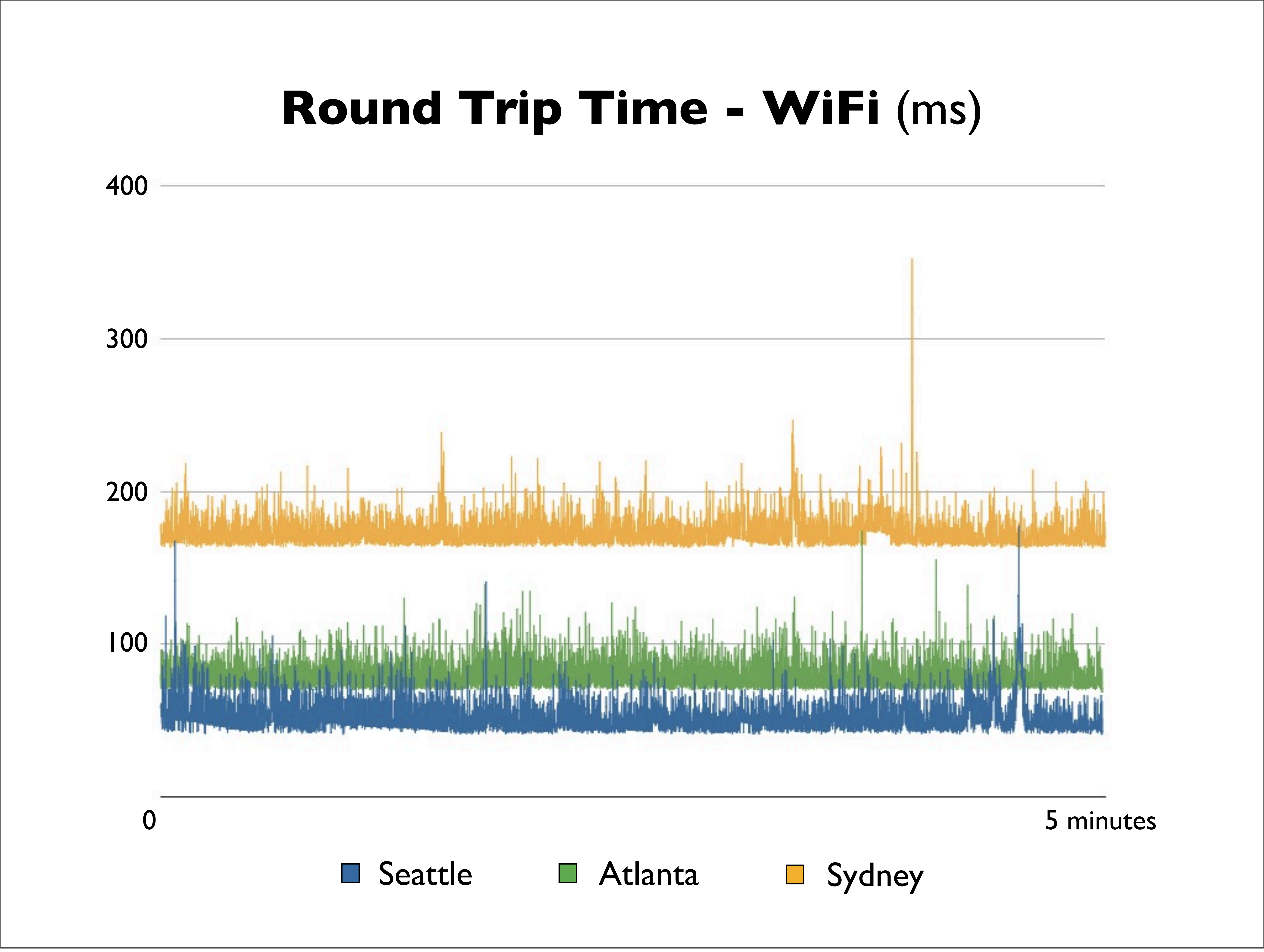
Zero lost packets for Seattle and Atlanta

Only **42** out of **9000** packets lost to Sydney and back. That’s less than 0.5% packet loss!

Jitter is reasonably large, also note the circled RTT spikes, what the hell... ?!

One standard deviation of packet RTT is:

10ms for Seattle
8.7ms for Atlanta
9.13ms for Sydney

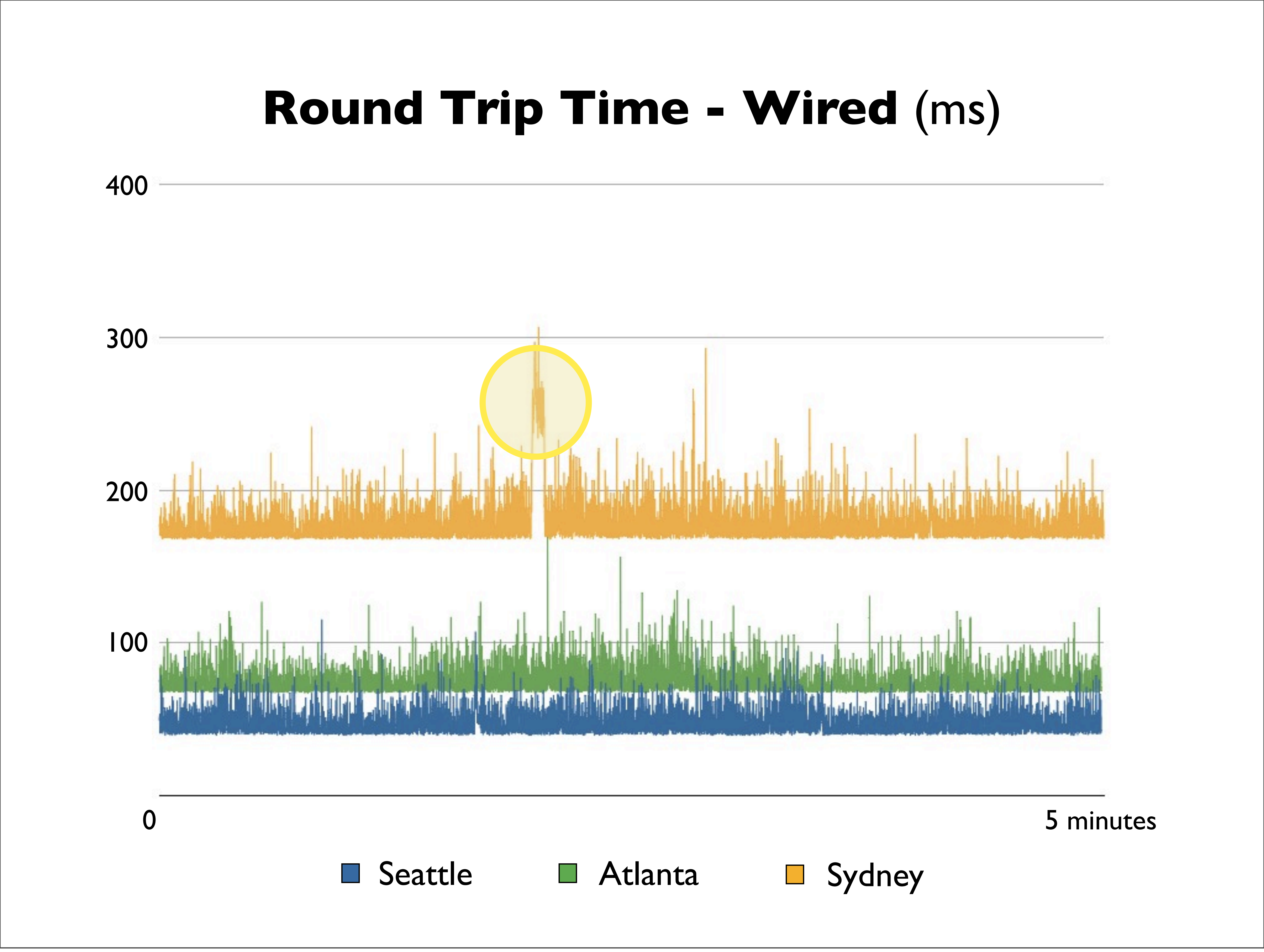


Round Trip Time - WiFi (ms)

Tuesday, March 16, 2010

Here is the detailed analysis output from my ruby script:

```
-----------------------------

  samples/wifi/seattle.txt:

    sample count = 9001
    max sequence = 9000
    lost packets = 0

    average RTT = 52.23
    standard deviation = 10.08

  samples/wednesday-evening/atlanta.txt:

    sample count = 9001
    max sequence = 9000
    lost packets = 0

    average RTT = 79.45
    standard deviation = 8.71

  samples/wifi/sydney.txt:

    sample count = 8959
    max sequence = 9000
    lost packets = 42

    average RTT = 173.05
    standard deviation = 9.13
```



Round Trip Time - Wired (ms)

Tuesday, March 16, 2010

So I thought OK, maybe the cat walked in front of the WiFi and caused the spikes... next time I try directly plugged in to my cable modem. Surely everything would be perfect now?

Not really.

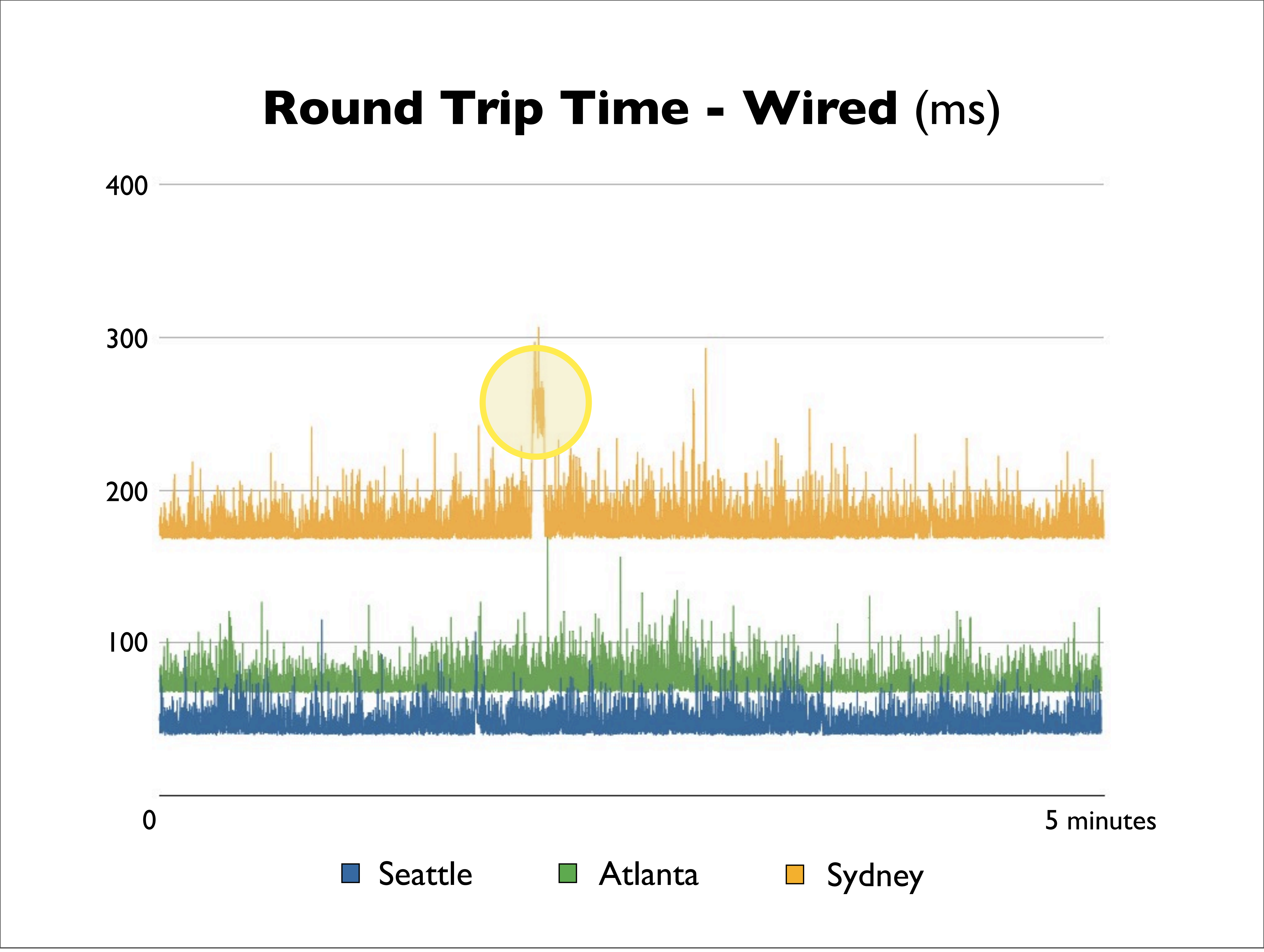
Seattle RTT went down from 53ms RTT to 47ms -- 6ms improvement

One standard deviation of jitter reduced from 10ms to 7ms -- 3ms improvement.

Only 23 packets lost to Australia and back, is this significant? Probably not.

Jitter to Sydney and back was actually worse. Definitely sources of jitter other than my WiFi.

Overall slightly better, but in roughly the same ballpark! -- note there is still a large sustained RTT spike in the Australia graph (?!)



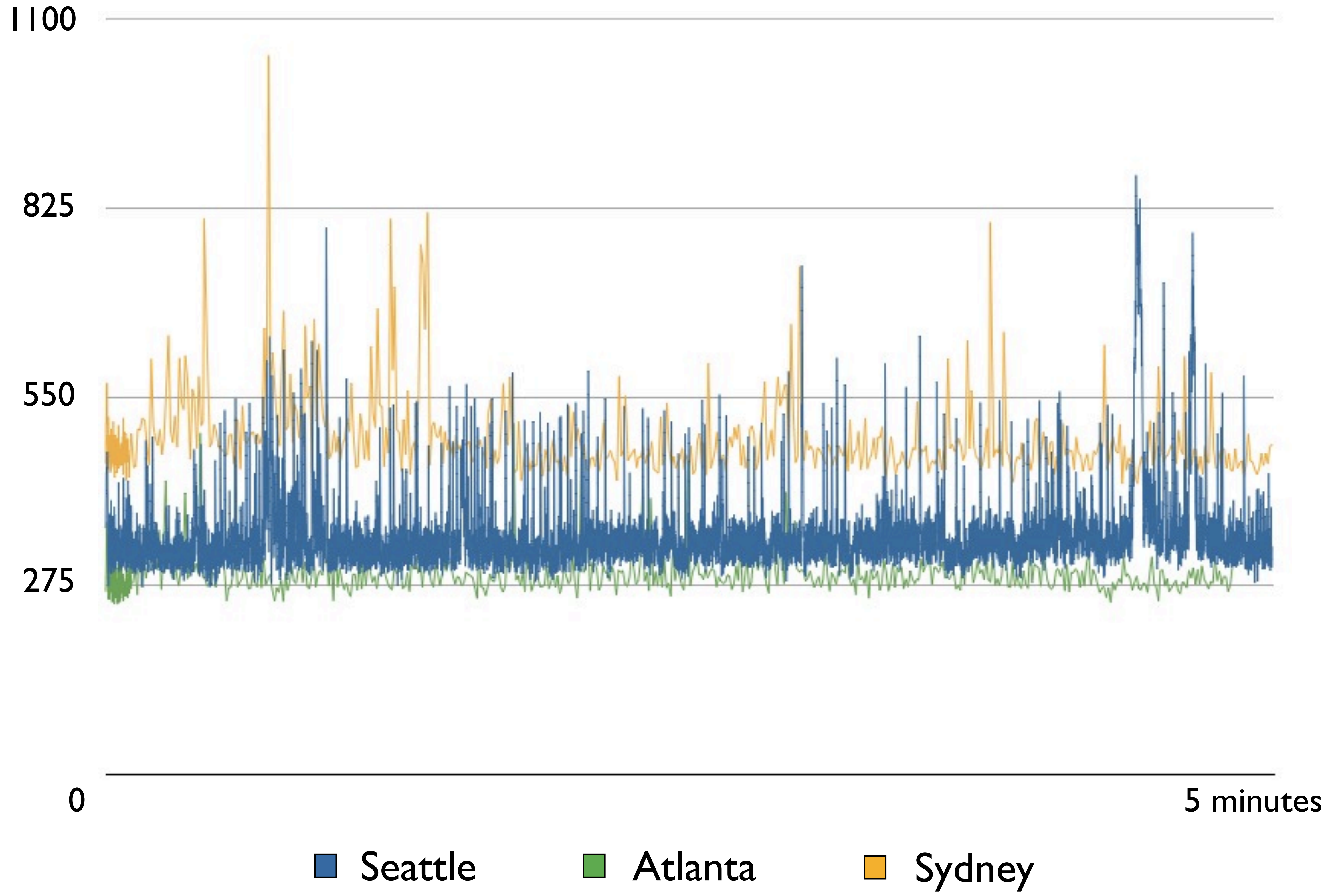
Round Trip Time - Wired (ms)

Tuesday, March 16, 2010

Here is the detailed analysis:

```
----------------------------

  samples/wired/seattle.txt:

    sample count = 9001
    max sequence = 9000
    lost packets = 0

    average RTT = 47.4134293270746
    standard deviation = 7.7117658130723

  samples/wired/atlanta.txt:

    sample count = 9001
    max sequence = 9000
    lost packets = 0

    average RTT = 74.5527052237528
    standard deviation = 8.1440094131561

  samples/wired/sydney.txt:

    sample count = 8978
    max sequence = 9000
    lost packets = 23

    average RTT = 177.743882071619
    standard deviation = 13.2487919391835
```

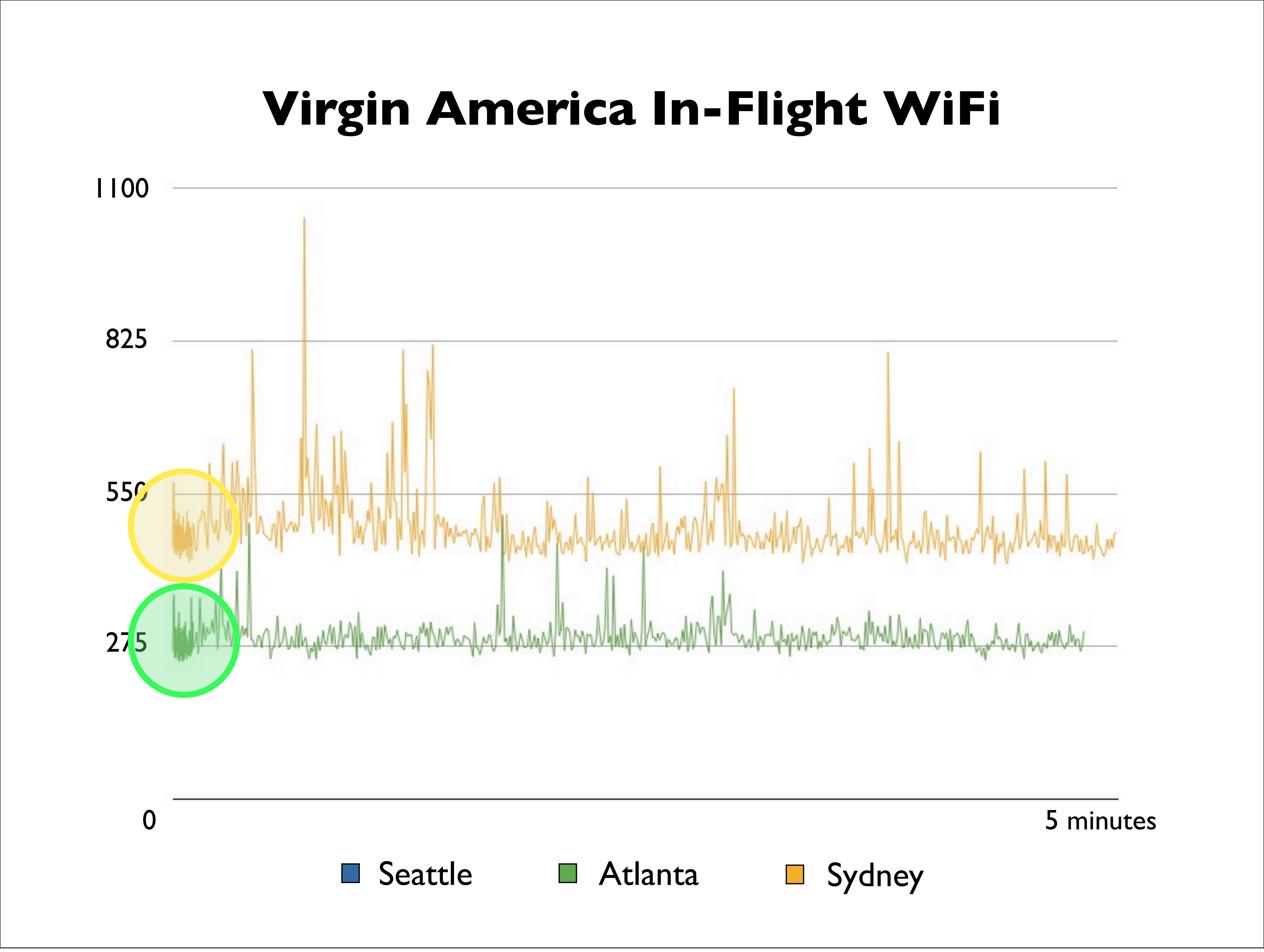
Virgin America In-Flight WiFi



Just for kicks I sampled RTT times on the flight over here using Virgin America’s WiFi service

Here are the results, note the super-high packet loss for Atlanta and Sydney -- I will explain on the next slide...

```
------------------------

  samples/virgin/seattle.txt:

    sample count = 8990
    max sequence = 9000
    lost packets = 11

    average RTT = 351.638138424027
    standard deviation = 61.2823643239847

  samples/virgin/atlanta.txt:

    sample count = 722
    max sequence = 8984
    lost packets = 8263

    average RTT = 289.63182949446
    standard deviation = 27.1364142427556

  samples/virgin/sydney.txt:

    sample count = 704
    max sequence = 8990
    lost packets = 8287

    average RTT = 485.126497289772
    standard deviation = 55.3614542536085
```



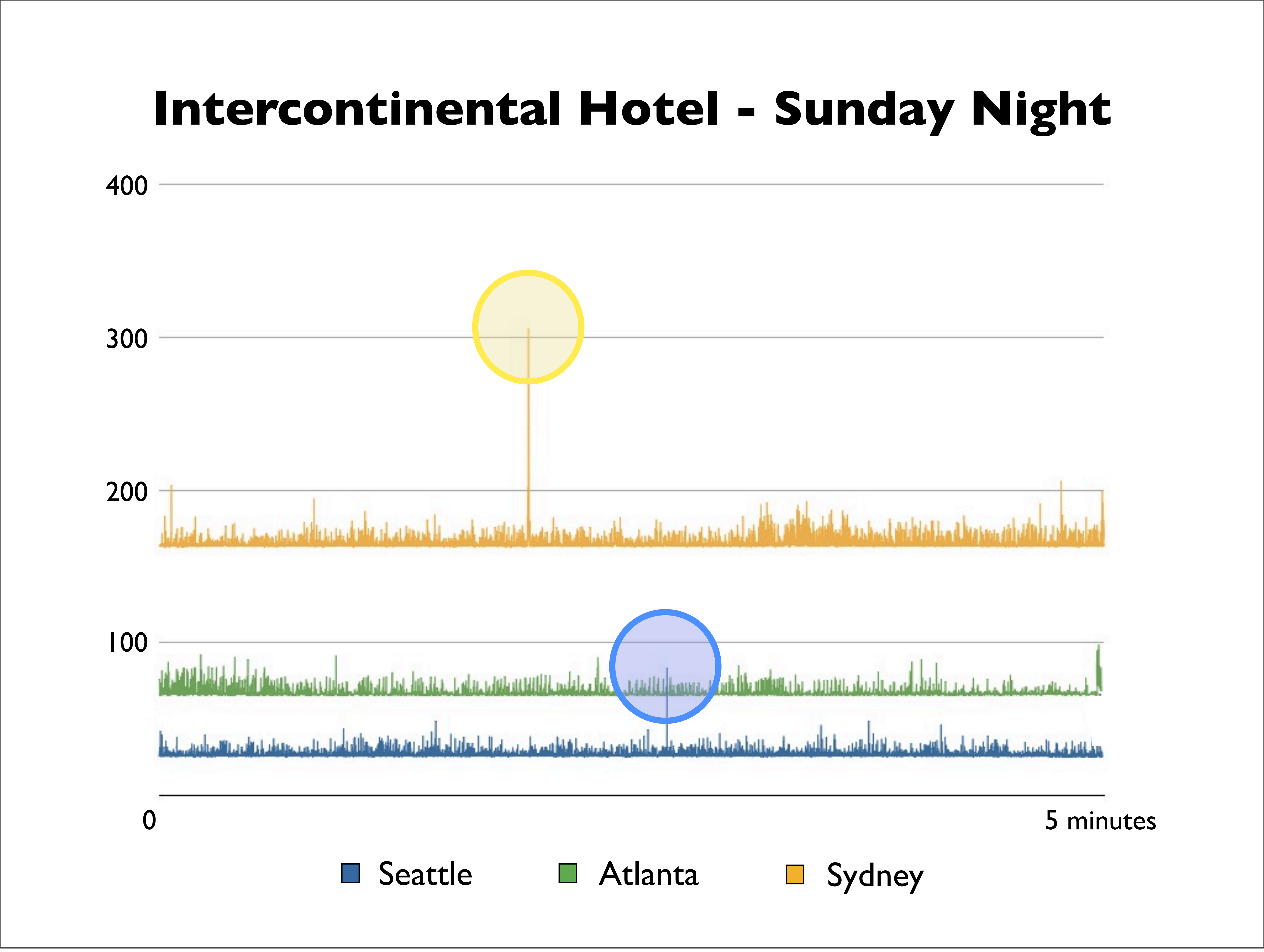
Virgin America In-Flight WiFi

Tuesday, March 16, 2010

Now with seattle hidden, look at the samples at the left. the router is shaping me like crazy -- each time i switch to a new IP it gives me a burst of a few seconds where 100% packets get through -- note the graph is really dense for the first bit on the left, then gets more sparse -- after this it only lets through one packet every second or so makes it, crazy! It’s almost 99% packet loss!

Ironically when I arrived in SF the Virgin Wireless sent me an email asking me “So what did you think of our WiFi service?”.

Hmmm! :)



Intercontinental Hotel - Sunday Night

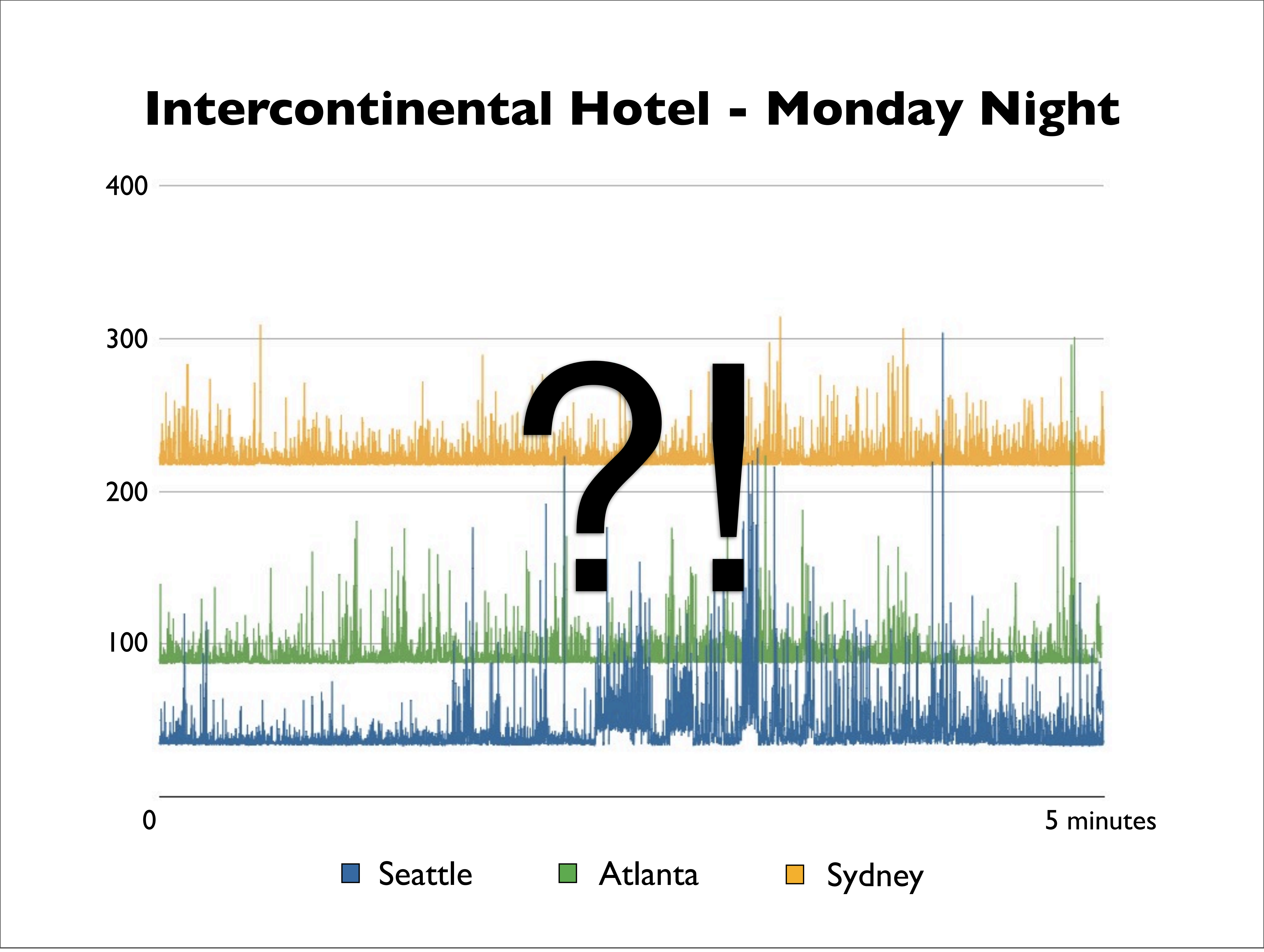
Tuesday, March 16, 2010

Upon arriving to SF of course I sampled my Hotel’s WiFi...

To my surprise it was actually the best RTT sample I’ve ever seen – WOW!

Very very flat RTT graph with just a few lonely spikes. It seemed too good to be true?!

```
  samples/intercontinental/seattle.txt:

    sample count = 8945
    max sequence = 9000
    lost packets = 56

    average RTT = 26.8950195913918
    standard deviation = 2.08503691203364

  samples/intercontinental/atlanta.txt:

    sample count = 8991
    max sequence = 9000
    lost packets = 10

    average RTT = 66.7807113194309
    standard deviation = 2.37586512211464

  samples/intercontinental/sydney.txt:

    sample count = 8933
    max sequence = 9000
    lost packets = 68

    average RTT = 165.041152326766
    standard deviation = 4.08977165189762
```



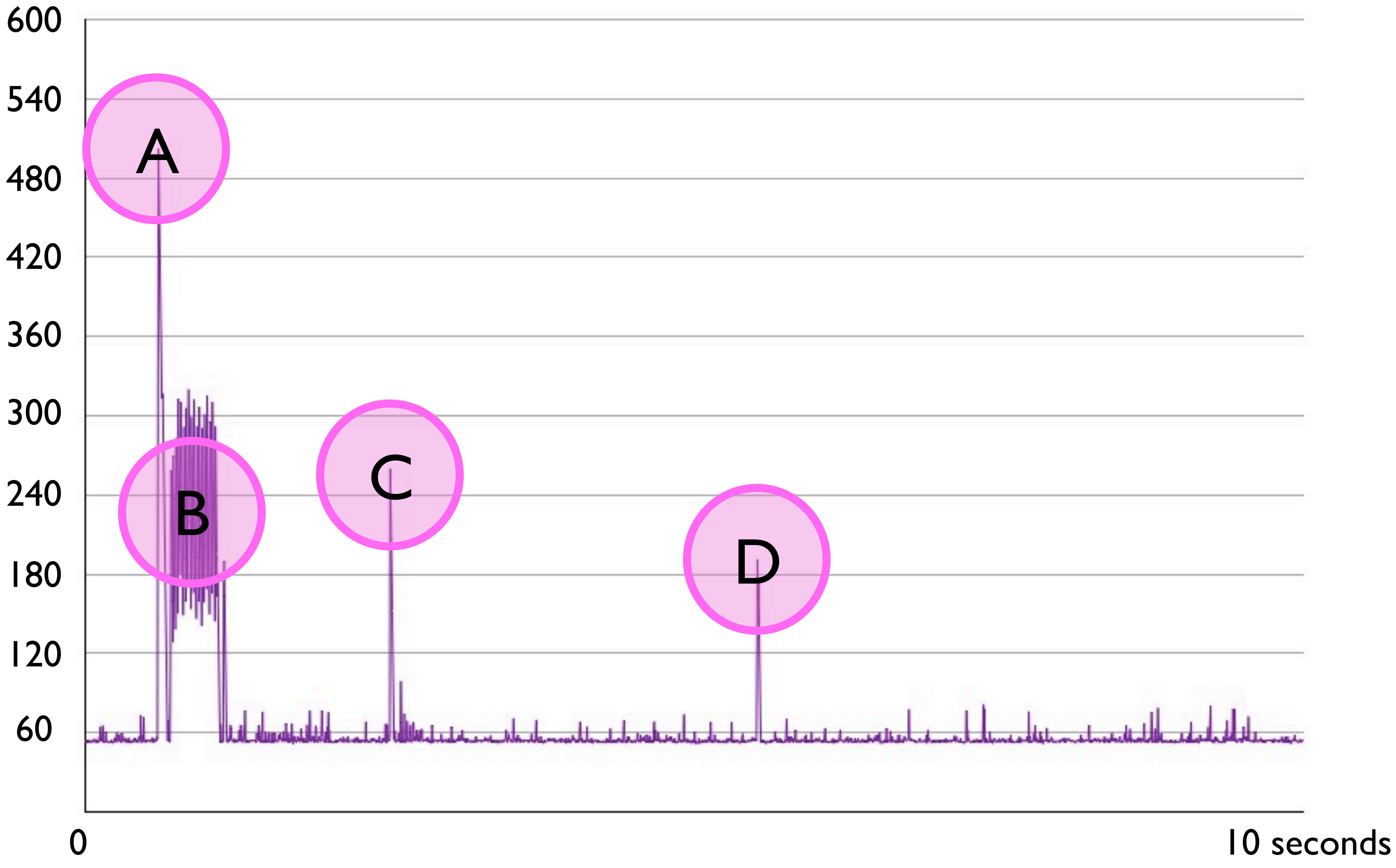
Intercontinental Hotel - Monday Night

So I tried it again the next day around the same and this is what I got -- WTF?!

GDC Nerd Invasion perhaps... really poor sample, especially look how spiky the seattle RTT is in the second half ...

```
------------

  samples/intercontinental-monday/seattle.txt:

    sample count = 8393
    max sequence = 9000
    lost packets = 608

    average RTT = 32.5382831620397
    standard deviation = 12.5023049232386

  samples/intercontinental-monday/atlanta.txt:

    sample count = 8477
    max sequence = 9000
    lost packets = 524

    average RTT = 69.2751138718886
    standard deviation = 7.29225816338481

  samples/intercontinental-monday/sydney.txt:

    sample count = 8548
    max sequence = 8999
    lost packets = 452

    average RTT = 166.2297241157
    standard deviation = 5.47549134232777
```

Typical RTT Spikes



To finally ram the point home.

Here are some a typical Round Trip Time (RTT) spike sampled over 10 seconds

This is not an average case, but perhaps an typical worst case -- what you can expect occasionally on a good connection or frequently, on a poor one :)

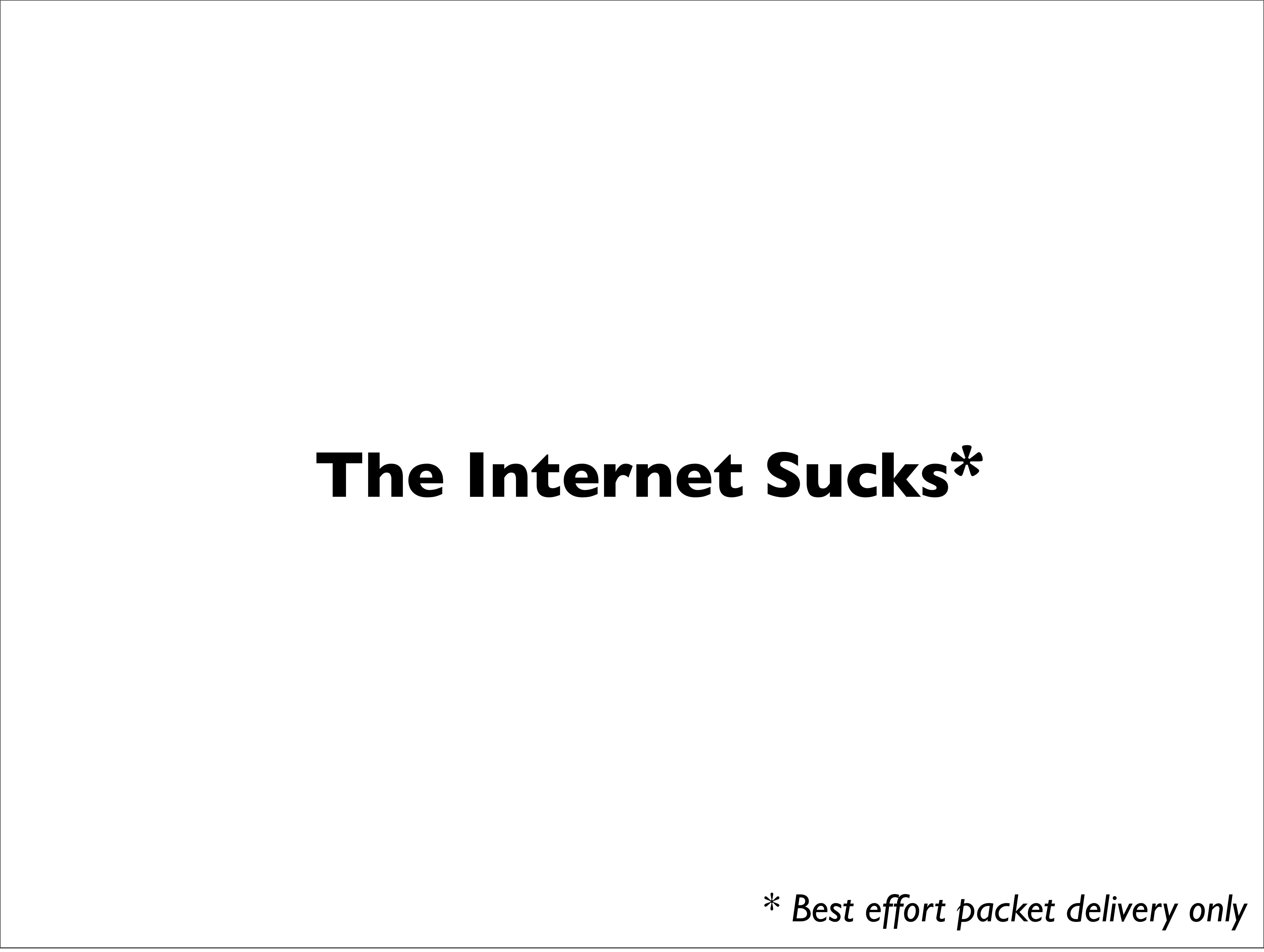
Notice spike A -- one packet just decides that it will arrive half a second late for no reason at all

B is a block of packets for half a second somewhere between 120 and 300ms -- compared with average RTT of ~60ms

C and D are some other late packet spikes

Note as well that in general the graph is has a small amount of baseline jitter -- most packets arrive between 60-80ms

This is the typical crap you have to deal with when developing an online game.



The Internet Sucks*

* Best effort packet delivery only

Tuesday, March 16, 2010

So it is now proven.

The internet sucks.

Why?

Because the internet is best effort only

it gives you absolutely no guarantee except that it will “try” to delivery the packet

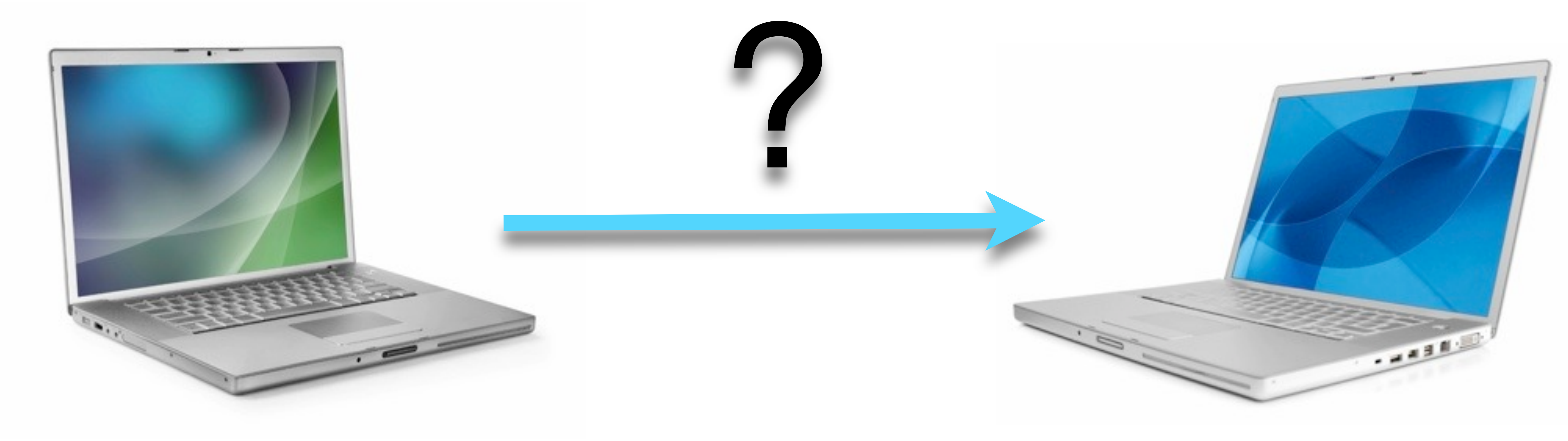
most importantly: the internet makes *no guarantee* that packets will arrive on time

recommendations:

- bandwidth: be conservative = 64kbit - 256kbit/sec (64k minspec)
- packet loss not too bad, better than you think, it’s actually quite rare
- latency is decent in general
- jitter and late packets are the key problem for realtime protocols (eg. voice, video, games etc...)

QoS issues
IPv6
flow routing
non-realtime “nice” bit in packet header
etc...

overall advice: be conservative with bandwidth usage, send rates, expectations of realtime packet delivery etc.



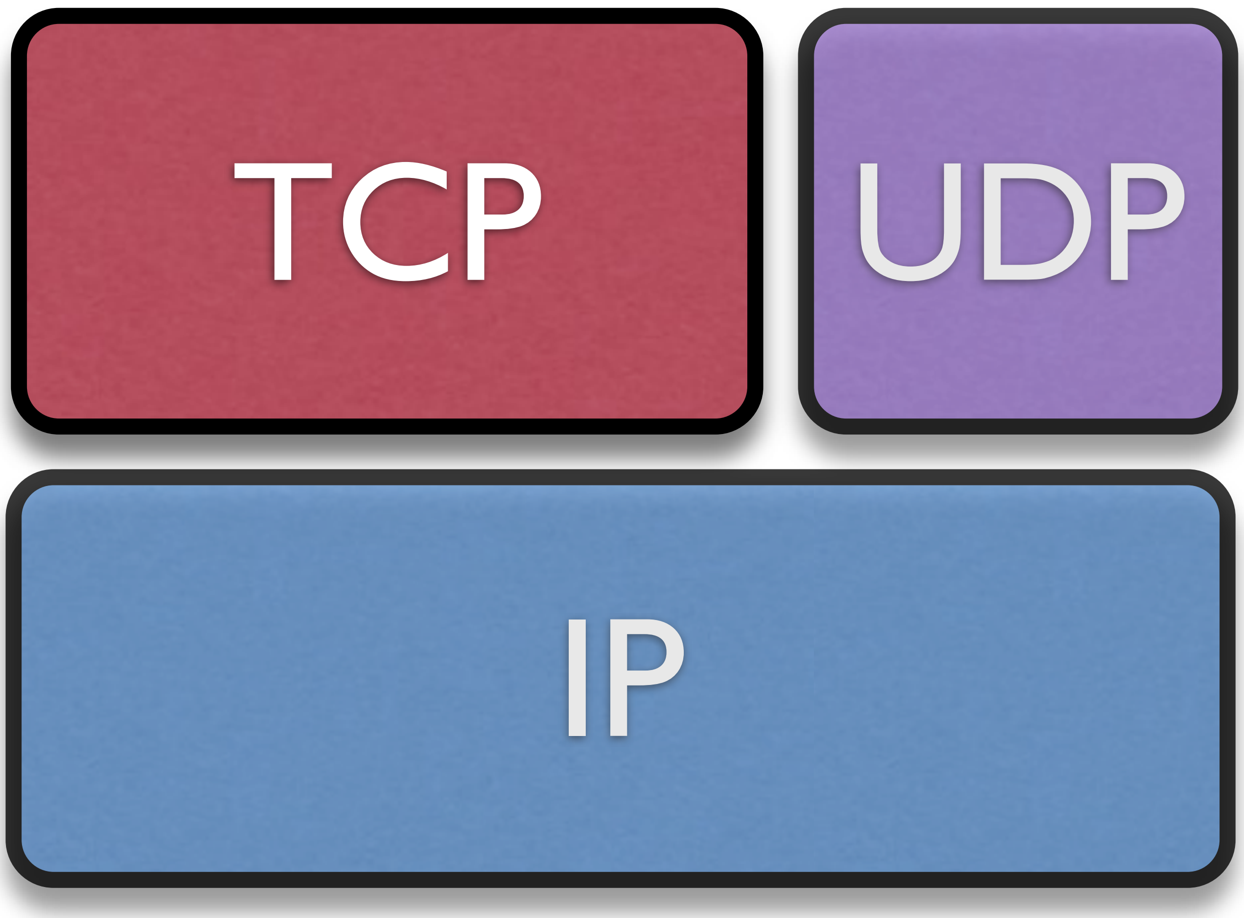
Tuesday, March 16, 2010

PART TWO

We have a simulation running on one computer -- how to view it on another?

Lets first look at how we will send packets between the computers...

We need to be very careful how we send our data, because it has been conclusively proven that the internet sucks



TCP

UDP

IP

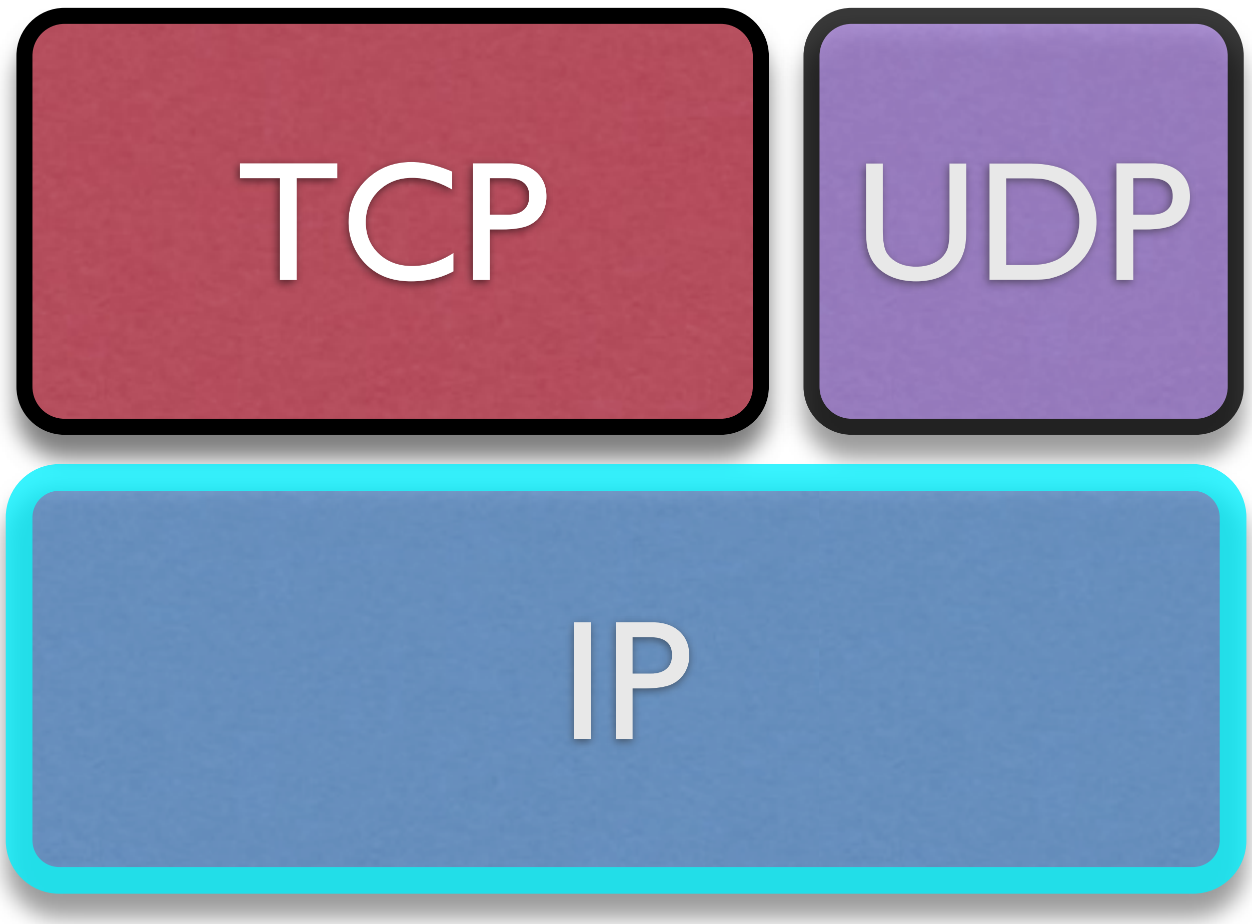
Tuesday, March 16, 2010

At the low level all packets travel over Internet Protocol (IP)

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are both built on top of IP.

Meaning that both UDP and TCP packets are actually sent inside IP packets.

Should we use TCP or UDP to synchronize our physics simulation?

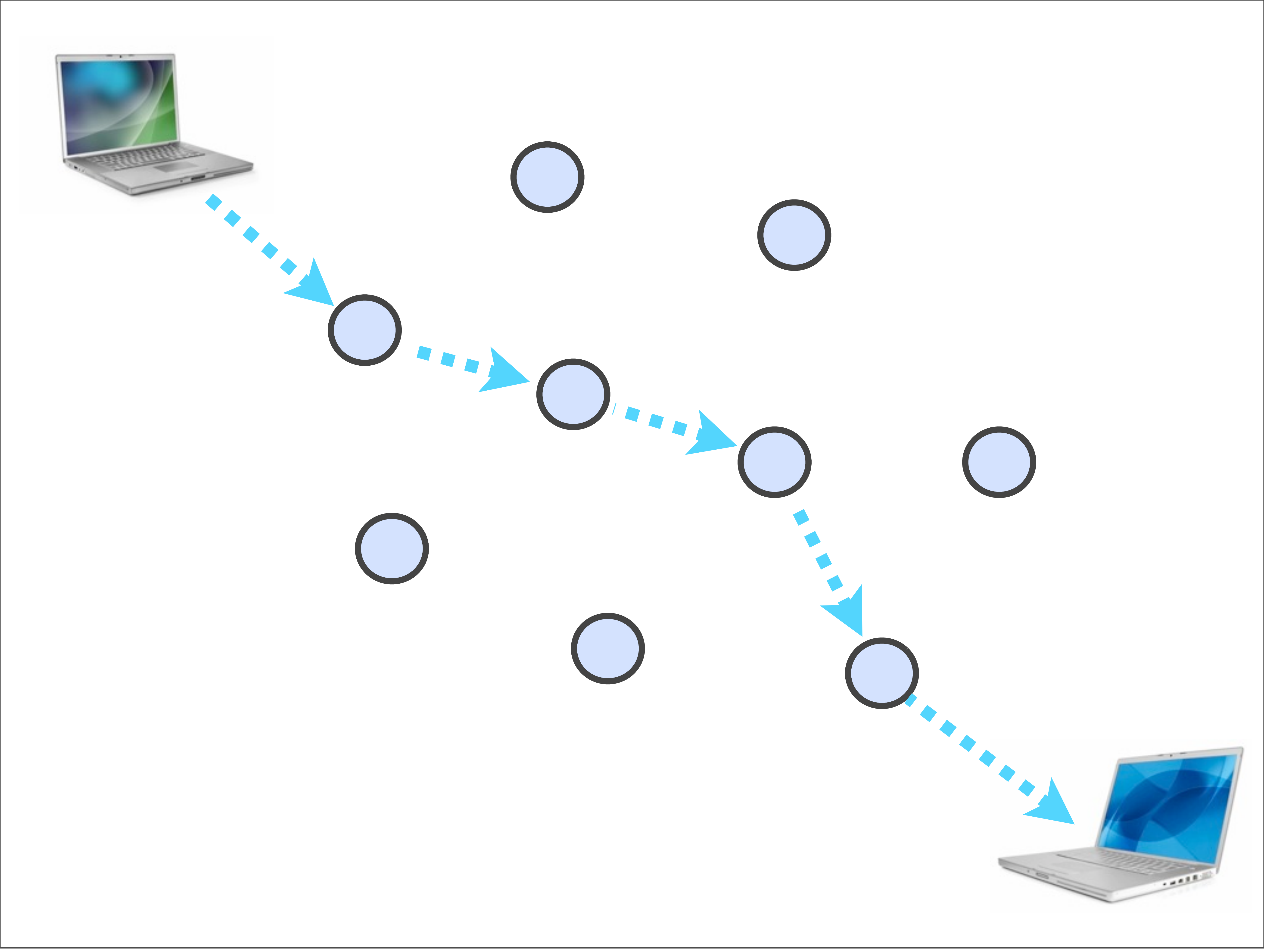


TCP

UDP

IP

To make the right decision we must first understand how IP works!

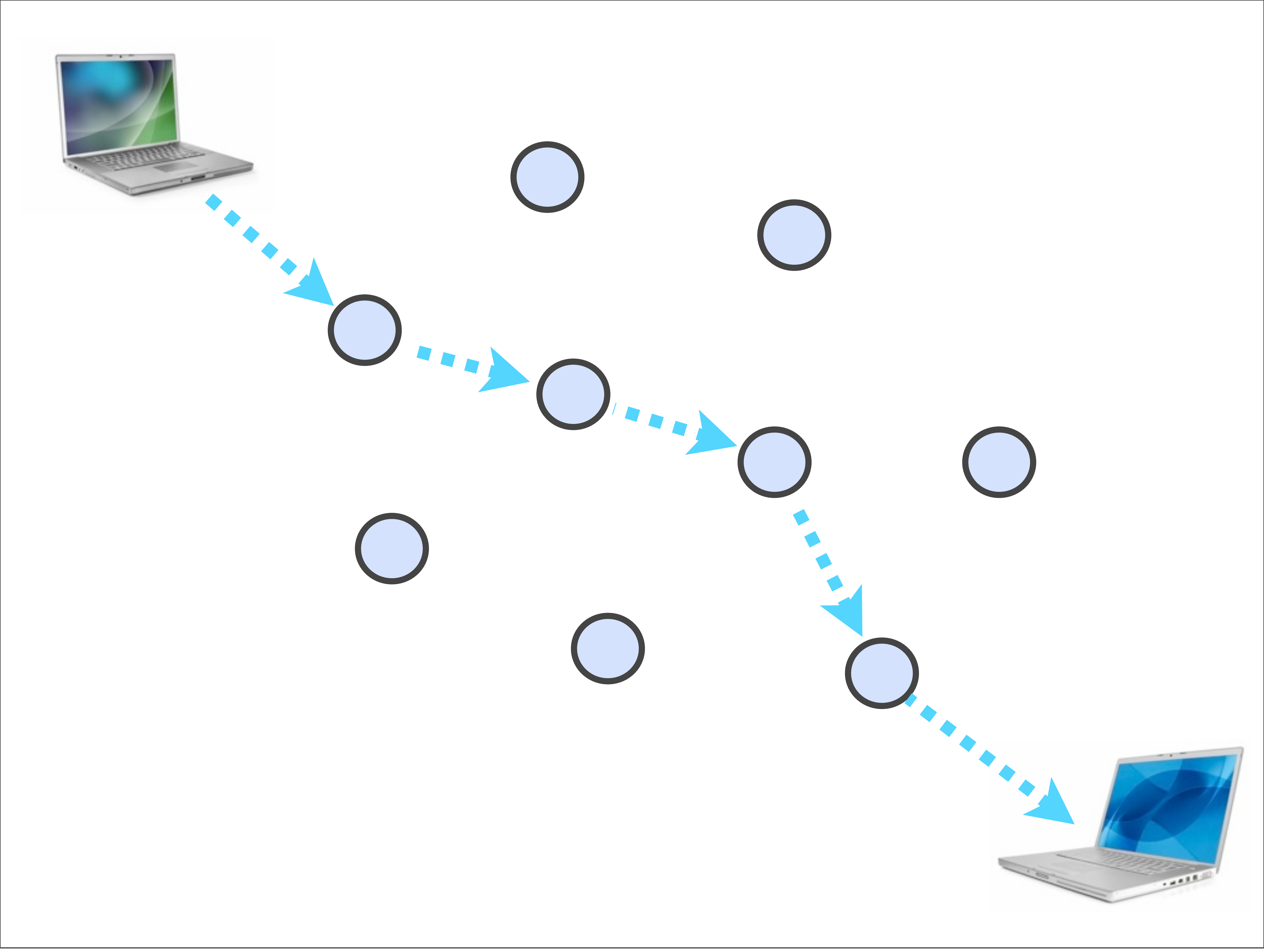


Tuesday, March 16, 2010

Here we have our little mini-internet

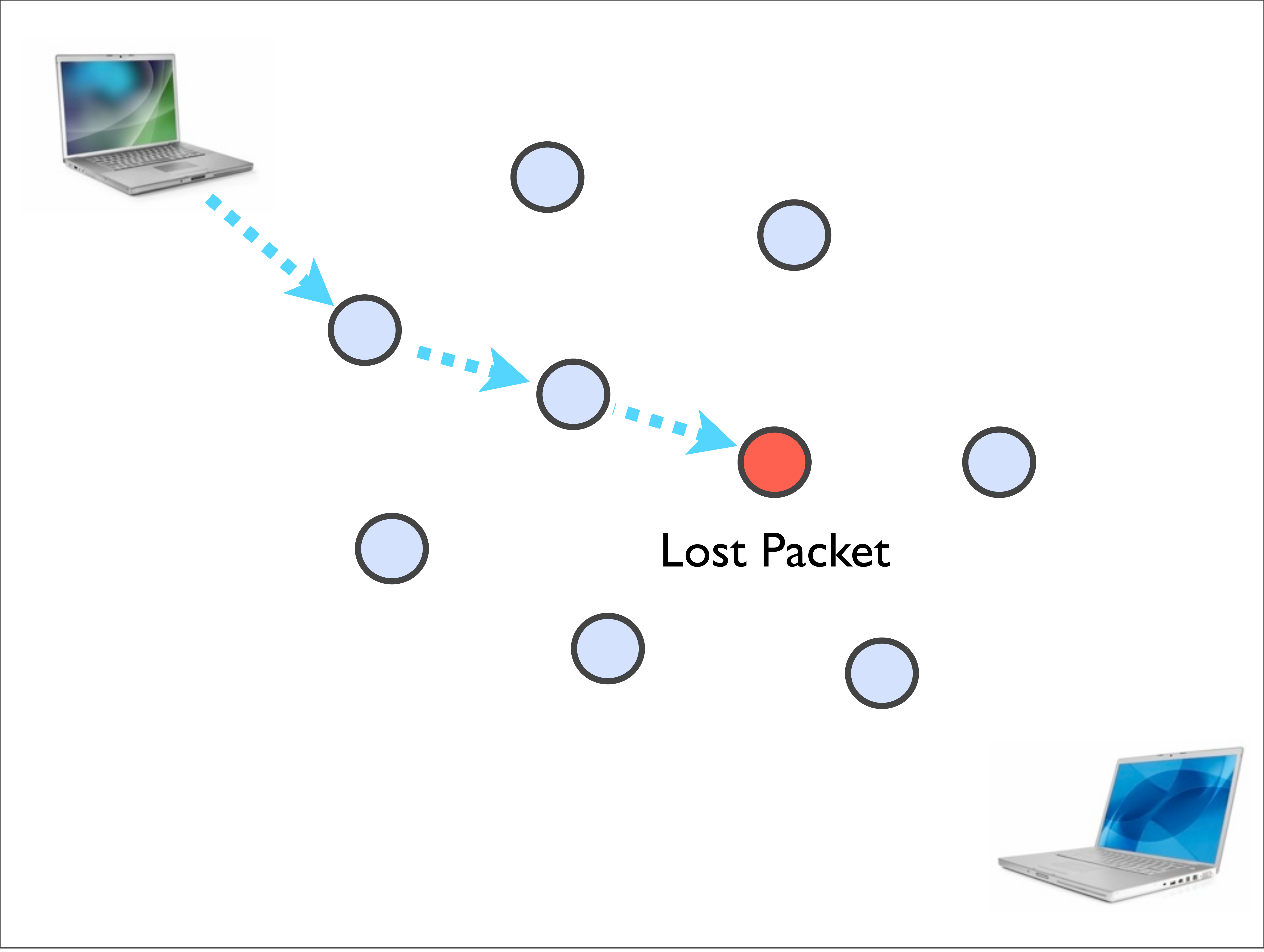
Each node between our two represents another node in the network like a computer or a router (same thing really...)

Packets are passed from node to node until they reach their destination



Each packet may take a different route to the destination.

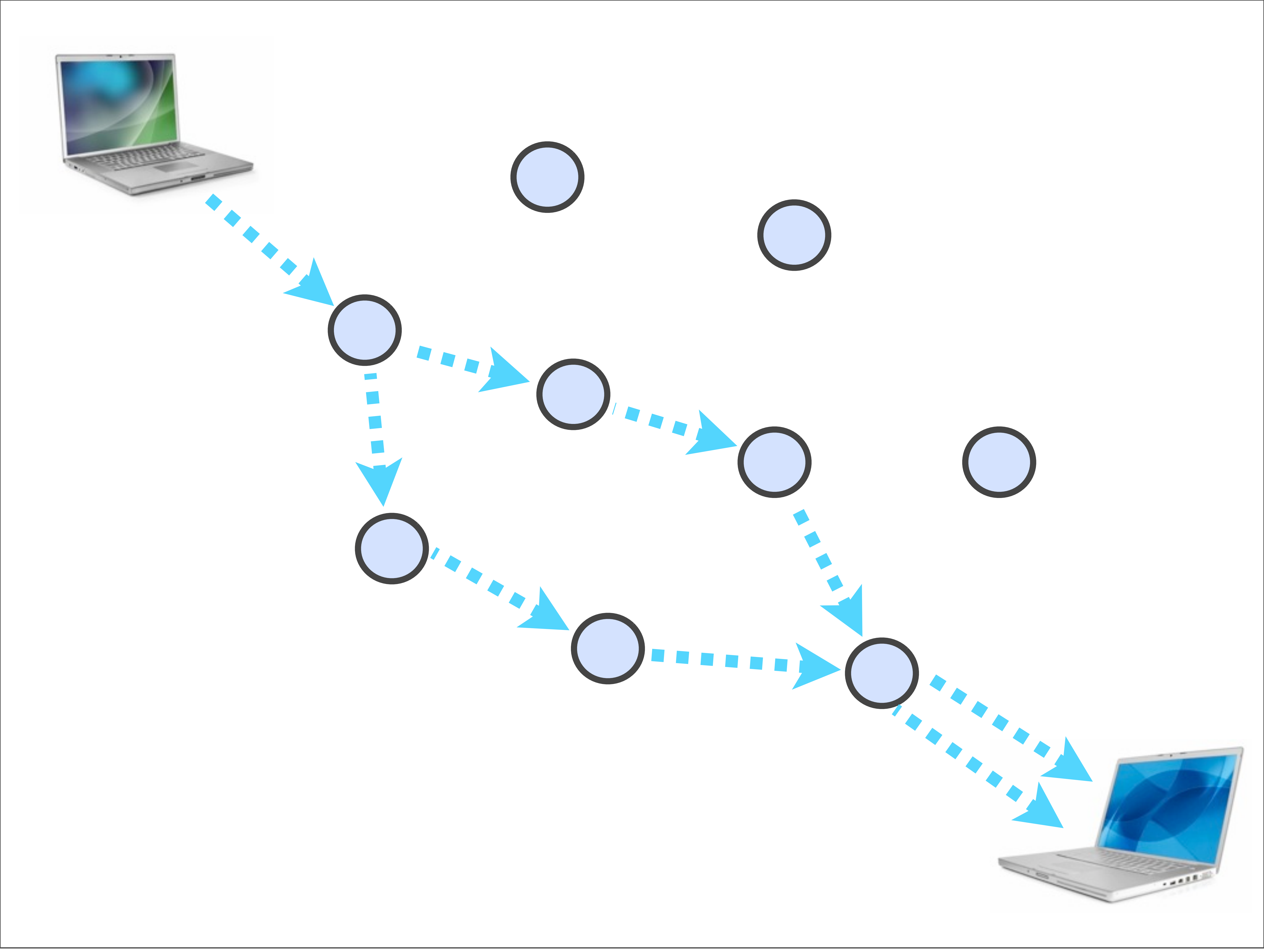
IP makes no guarantee that a number of packets sent in a row will all take the same route, even though this may occur.



Tuesday, March 16, 2010

Packets may be lost.

The sender and receiver are not notified when this happens. The packet just “disappears”.



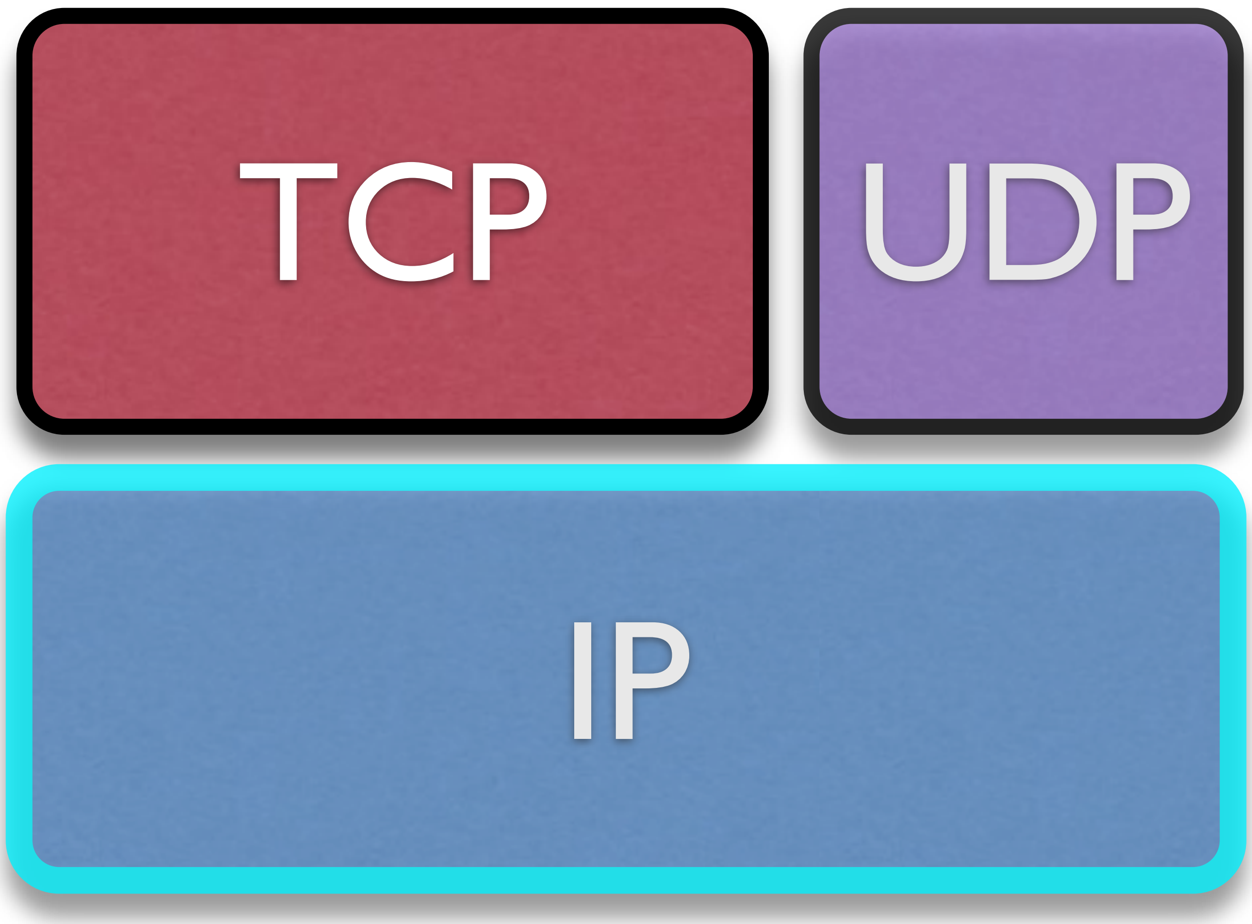
Tuesday, March 16, 2010

Packets may also be duplicated across multiple routes.

It also possible for one packet to take a much longer route than other packets.

These two things are the root cause of out of order and duplicate packets.

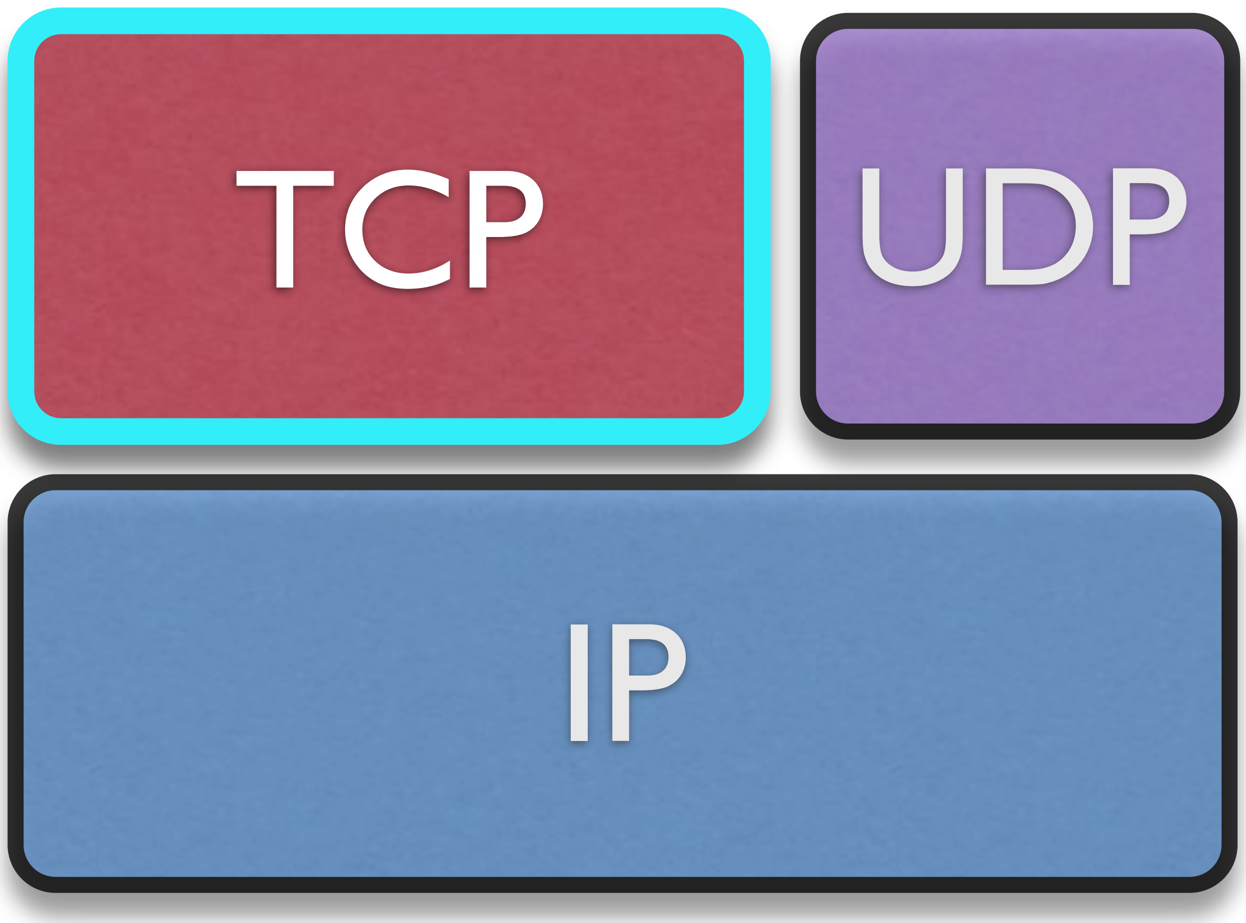
(What typically happens when the stable route changes in the middle of a sequence of packets you will temporarily see two streams of packets arriving from different routes, overlaid on top of each other until the route switch is complete.)



Tuesday, March 16, 2010

It is important to note that IP is how the internet really works

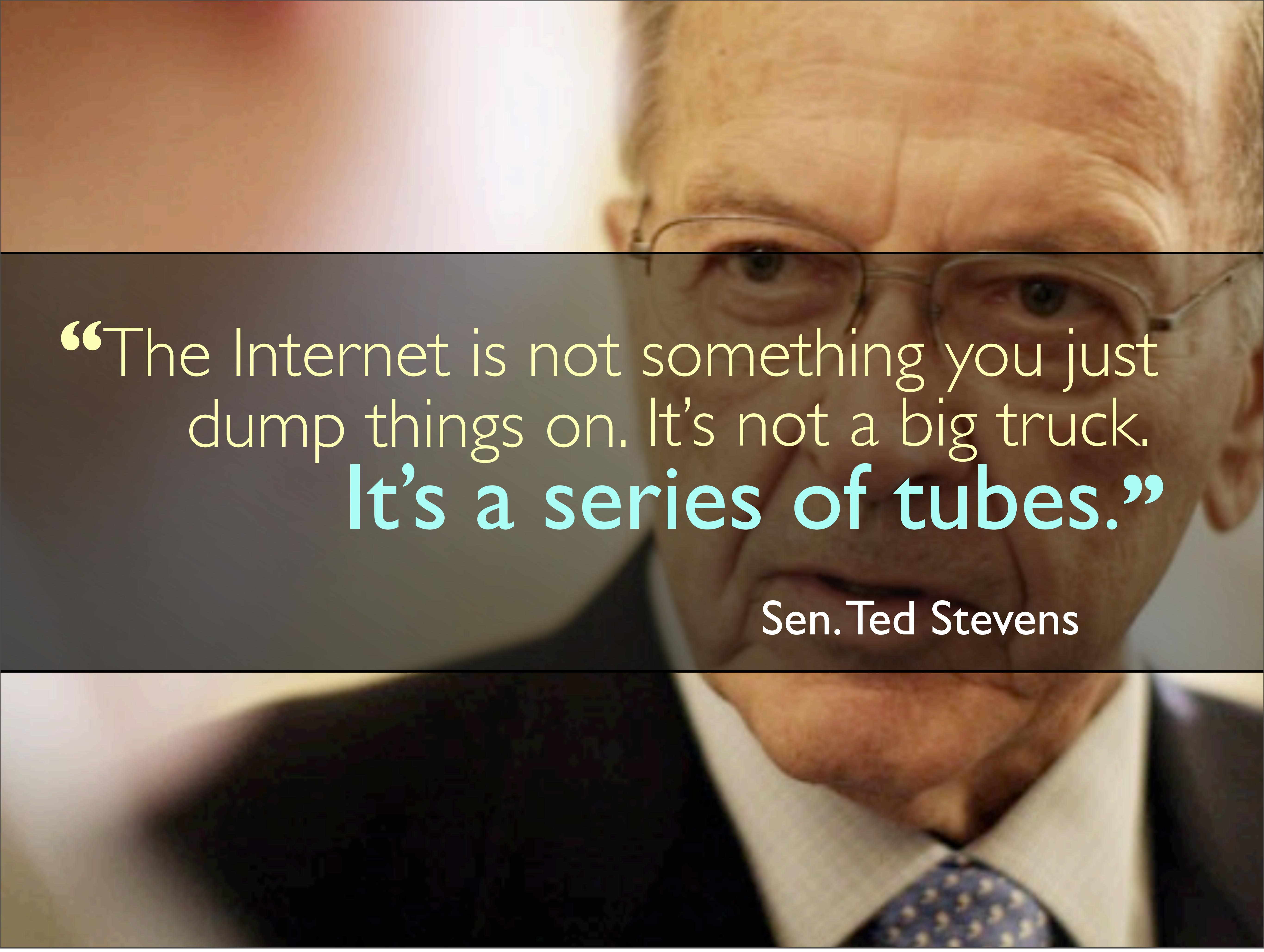
Both TCP and UDP work on top of IP, so they must both work within this model



If you have used TCP sockets then you know that TCP makes reading and writing data over the network just like reading and writing to a file

TCP guarantees that data arrives reliably and in-order

The internet does not work this way!

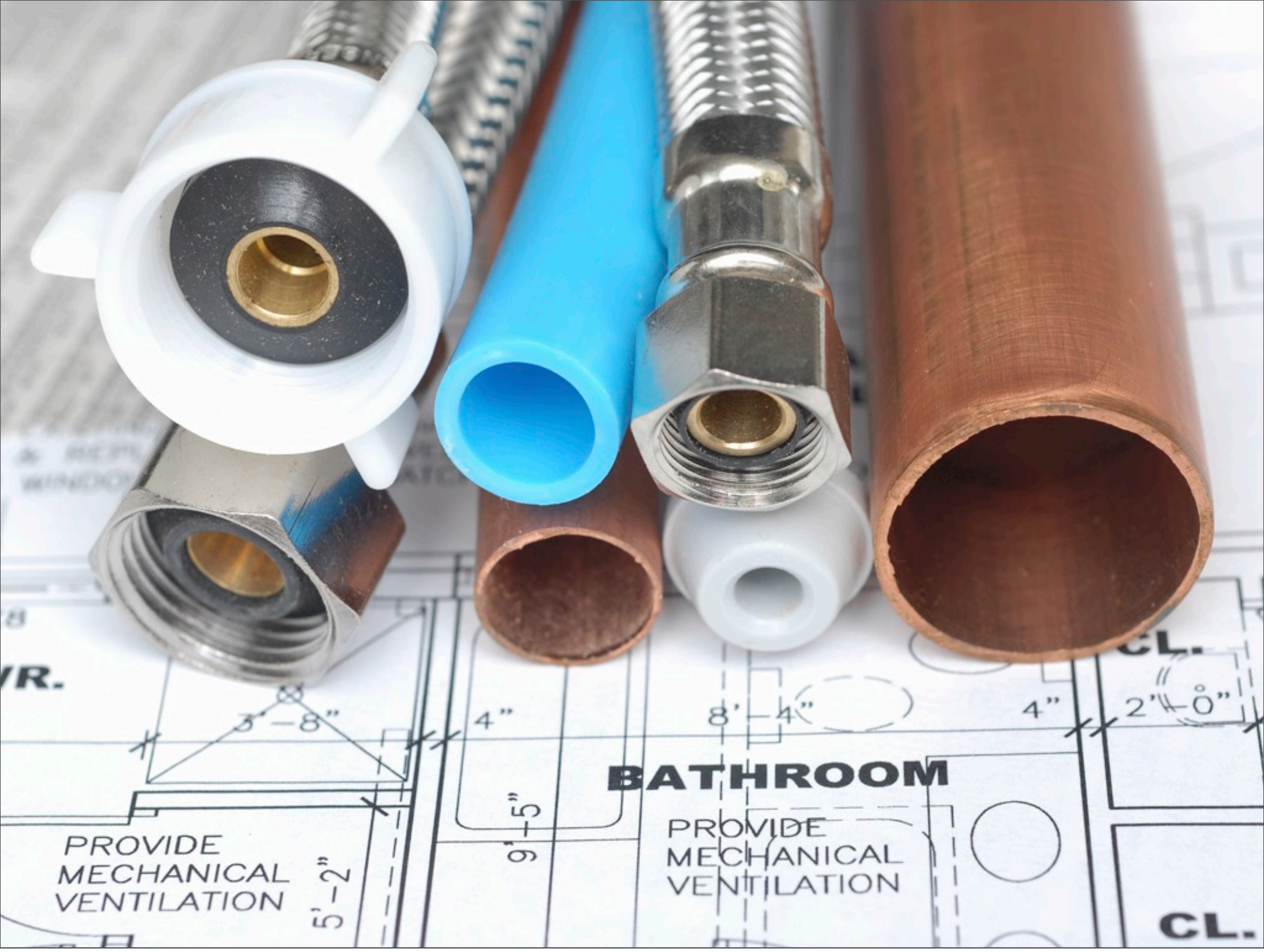


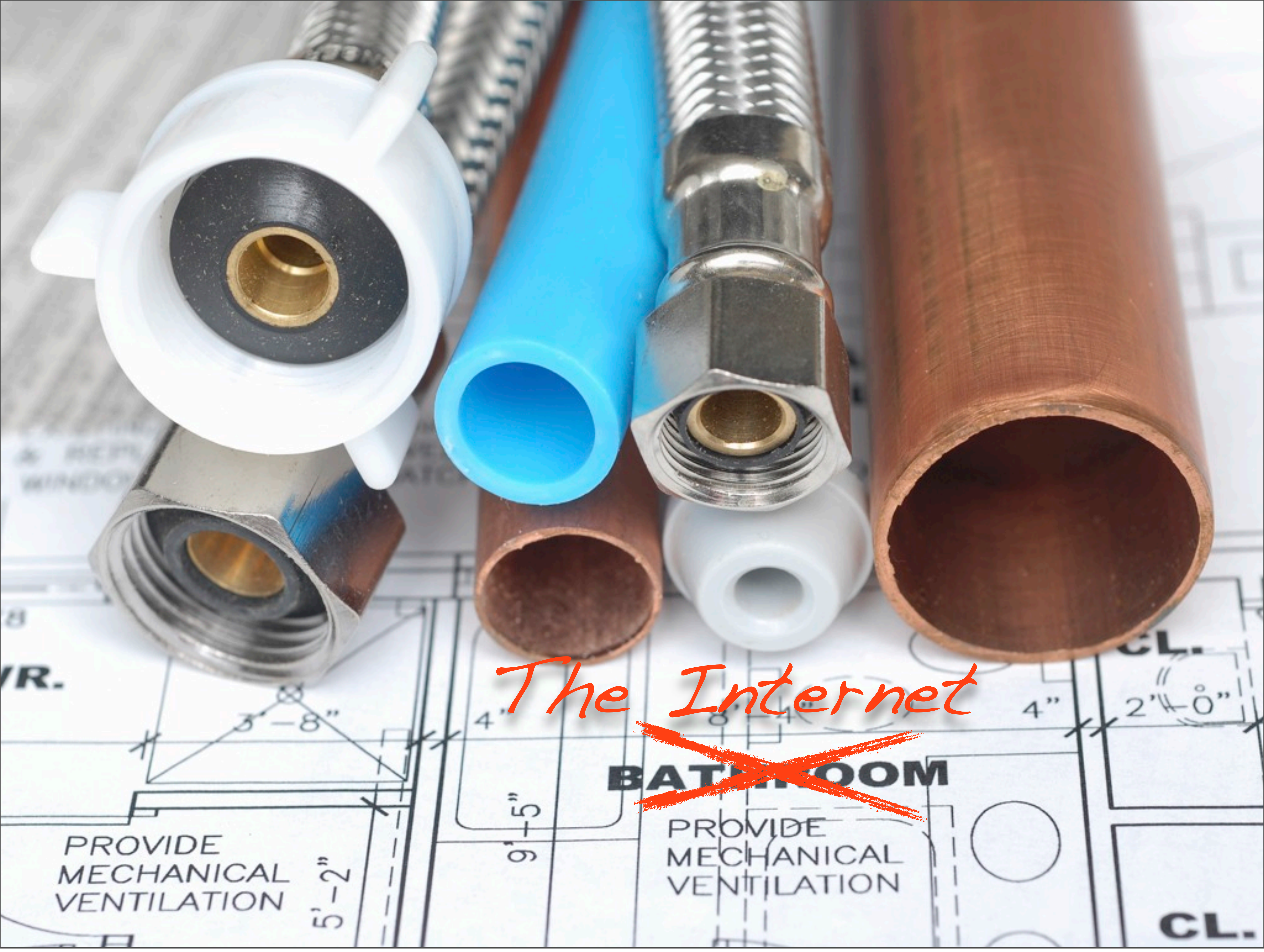
"The Internet is not something you just dump things on. It's not a big truck. It's a series of tubes."

Sen. Ted Stevens

Tuesday, March 16, 2010

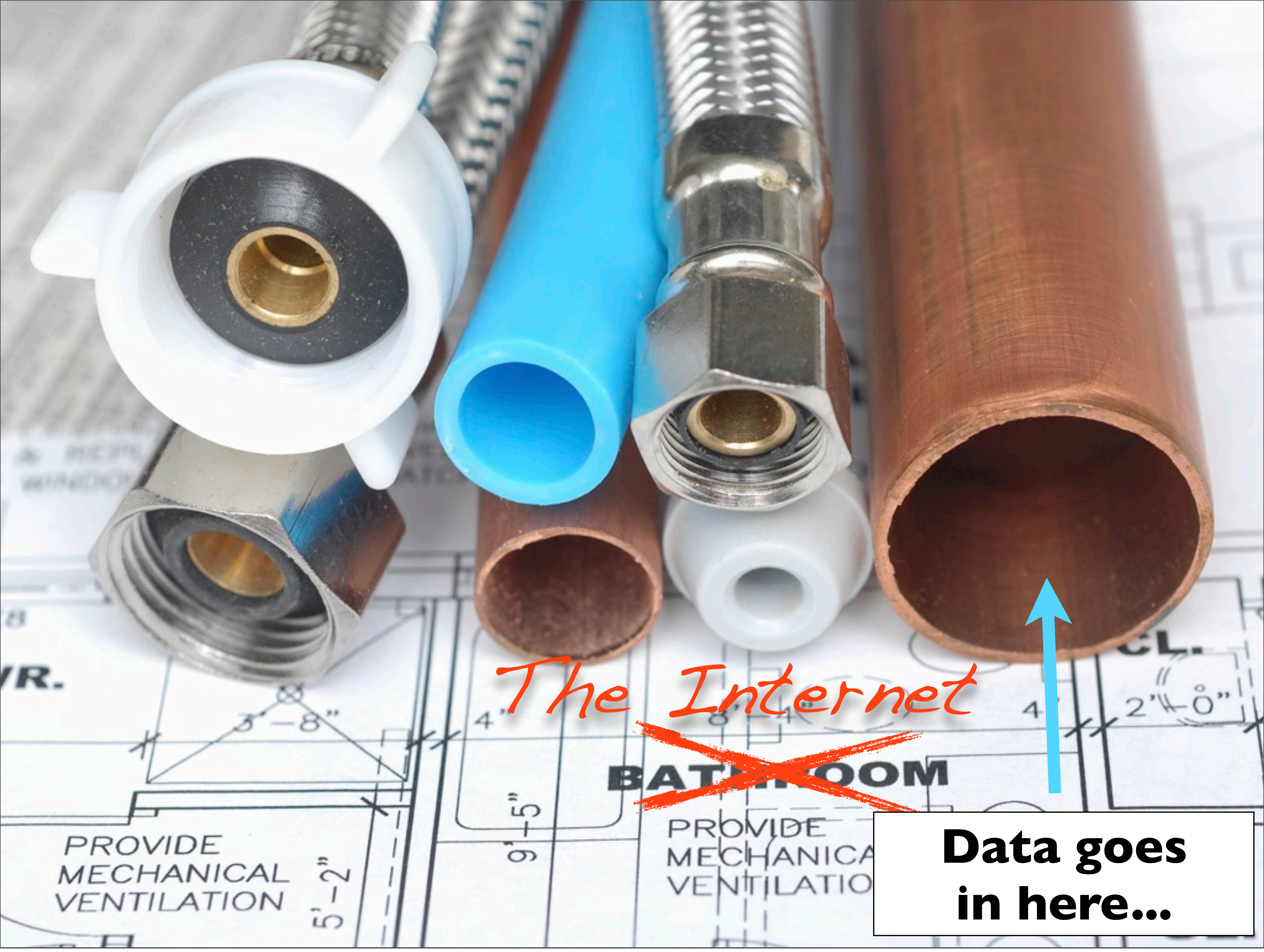
Unfortunately some people are a bit confused and think the internet works like TCP.


VR.
3'–8"
4"
8'–4"
4"
2'–0"
CL.
BATHROOM
9'–5"
PROVIDE
MECHANICAL
VENTILATION
5'–2"
PROVIDE
MECHANICAL
VENTILATION
CL.



The Internet

~~BATHROOM~~
PROVIDE MECHANICAL VENTILATION
PROVIDE MECHANICAL VENTILATION
CL.
CL.
3'–8"
4"
4"
9'–5"
5'–2"
2'–0"



The Internet

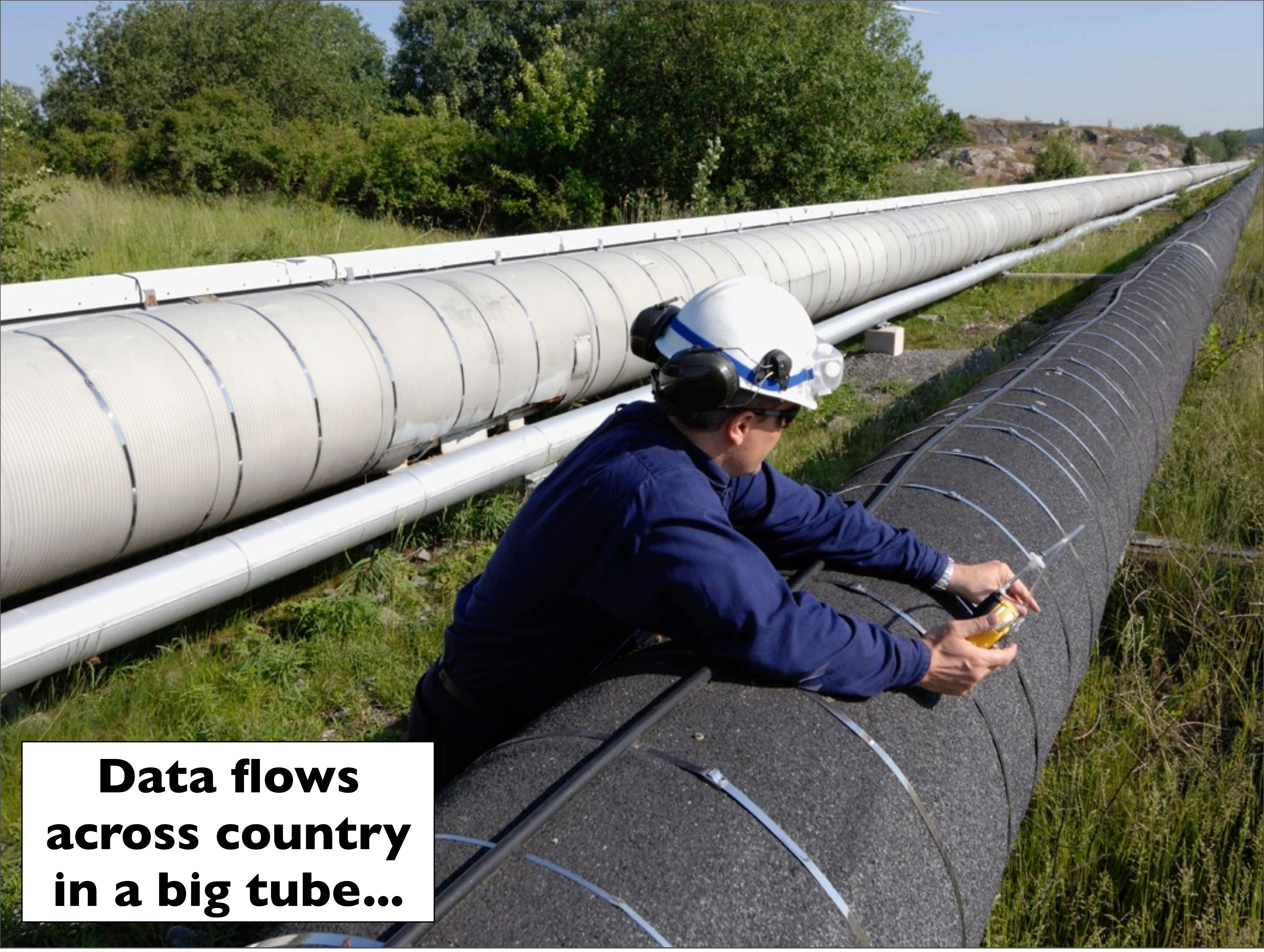
Data goes in here...

Data leaves your house...

Tube gets bigger...

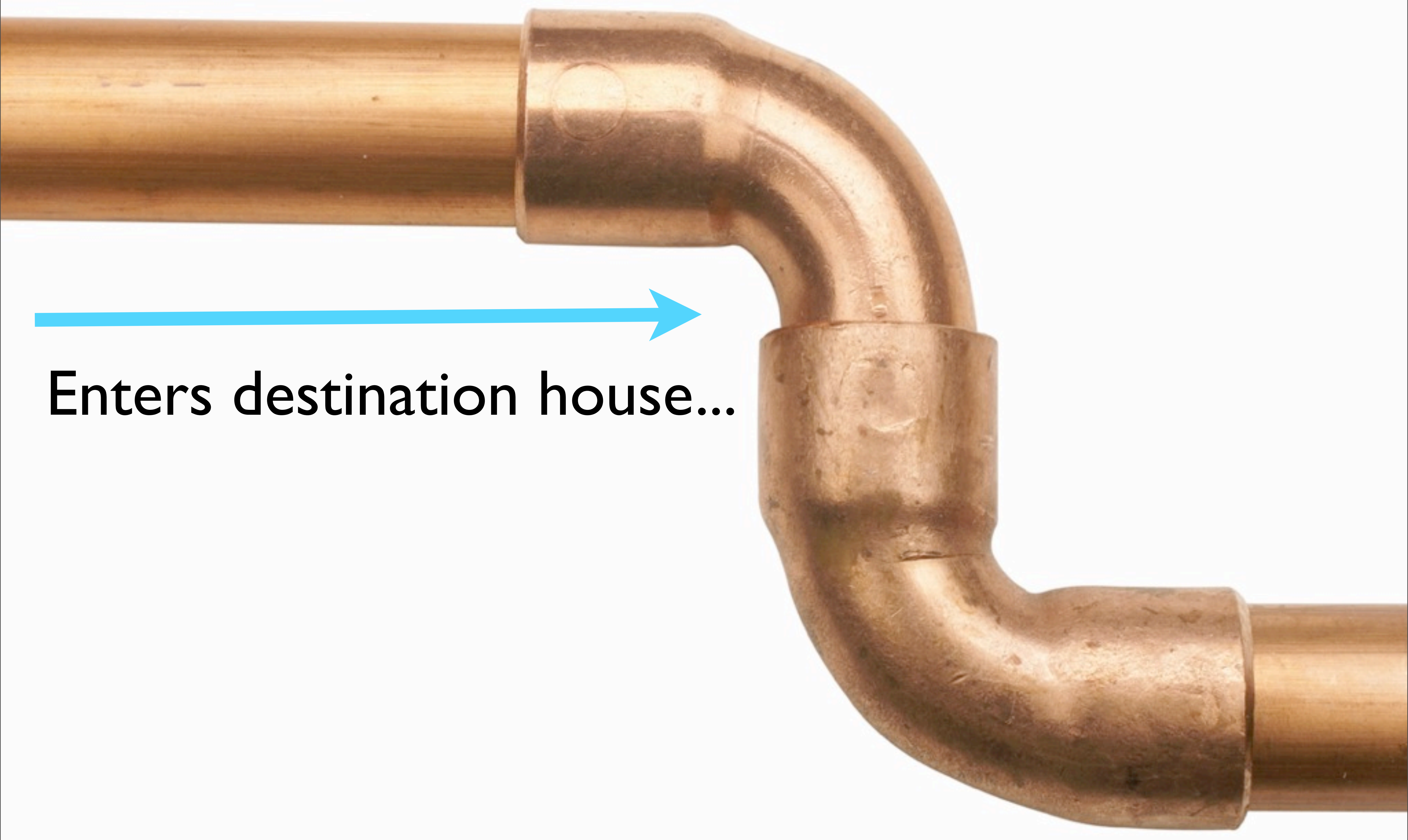
Data arrives at your ISP...

Lots and lots of tubes...

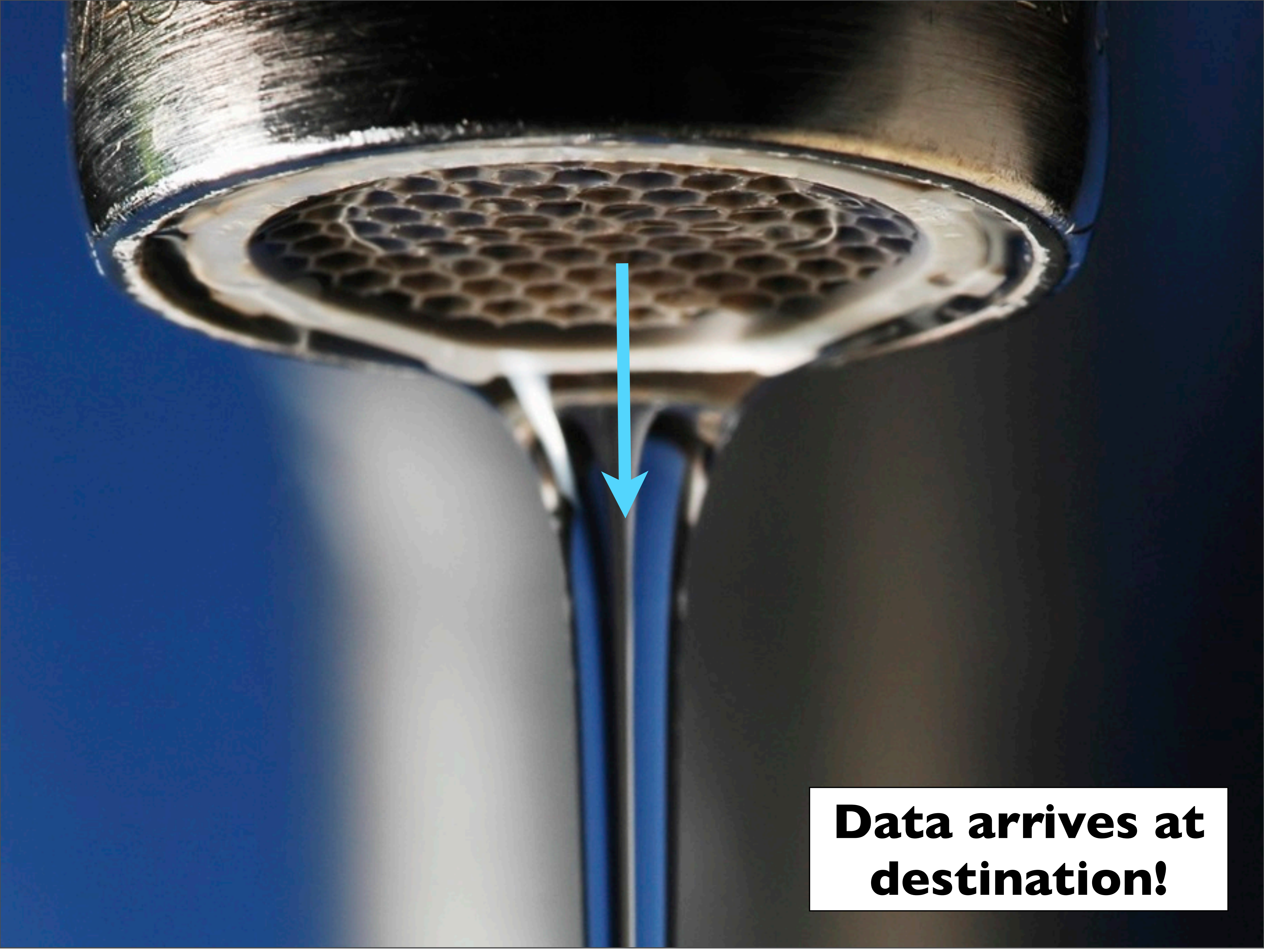


Data flows across country in a big tube...

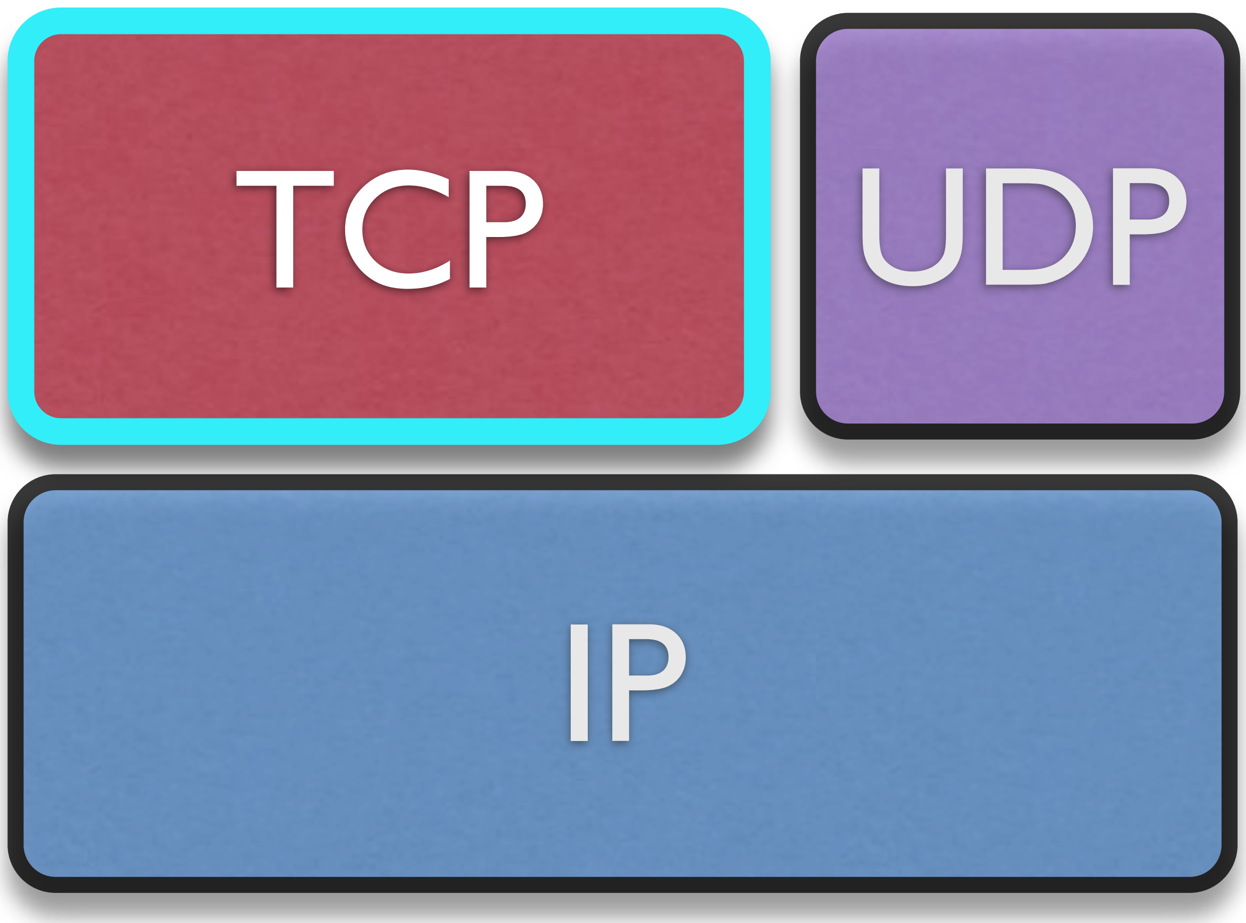
Tube gets smaller



Enters destination house...



Data arrives at destination!



TCP

UDP

IP

Tuesday, March 16, 2010

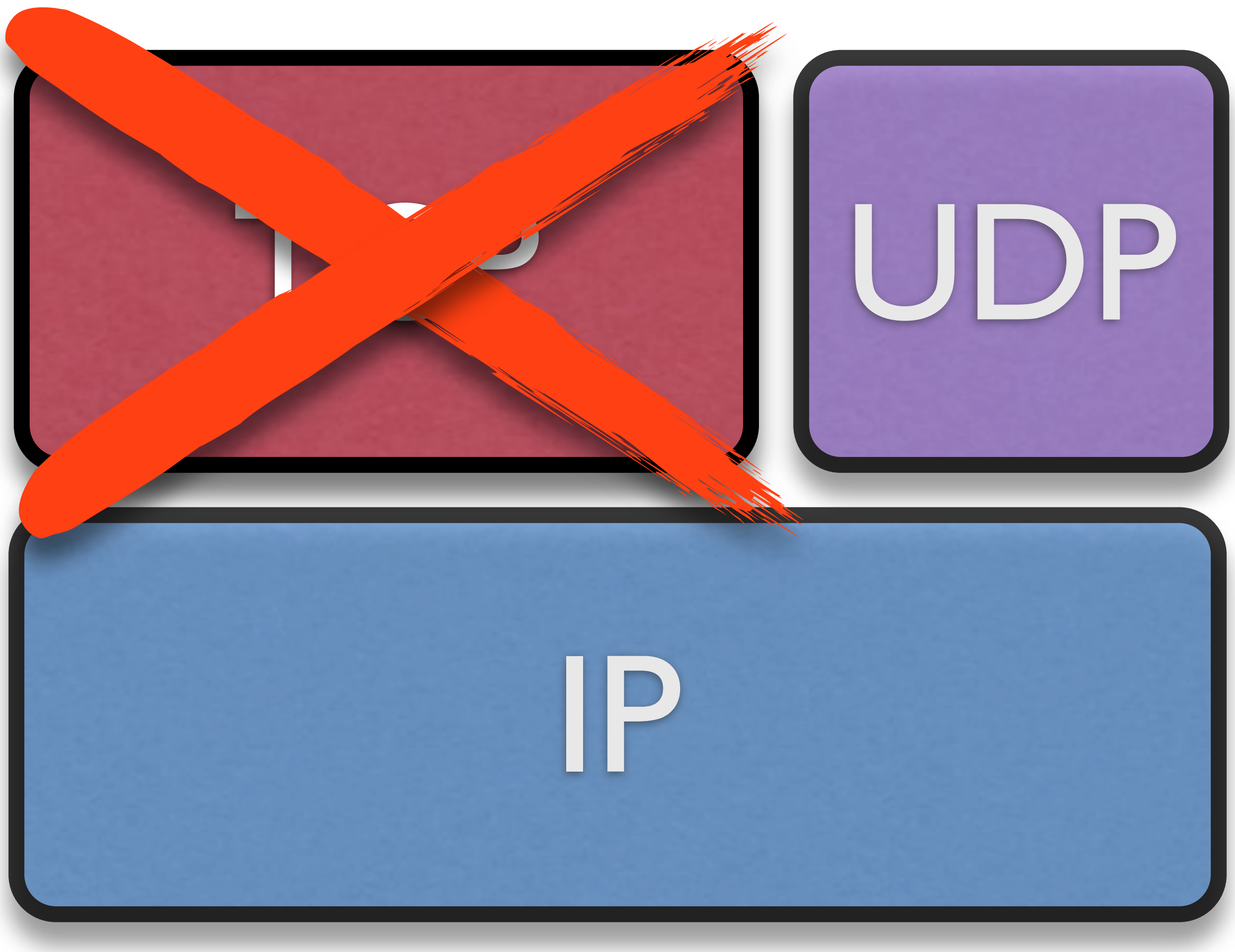
The internet does not work like this

The internet is packet based

The internet has limited bandwidth.

The internet does not guarantee that your packets will arrive...

Let alone that they will arrive at nicely spaced intervals on the other side – some packets will be very late!



~~TCP~~

UDP

IP

The most important part of this talk: **Never use TCP for realtime protocols!**

Why?

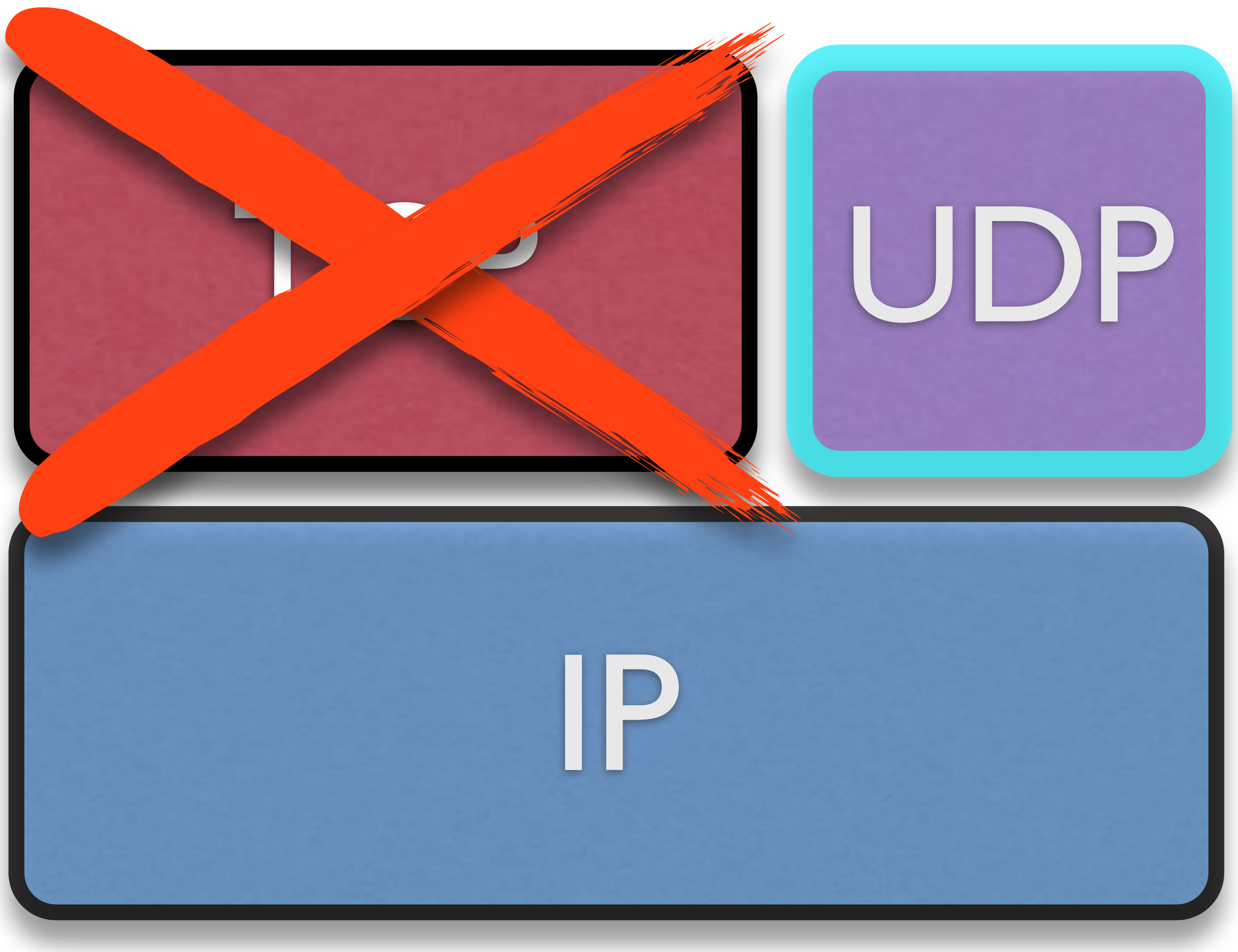
Because TCP delivers packets reliably and in-order the receiver must wait for lost packets before receiving most recent packets.

We do not want this. We want the most recent packet as soon as it arrives!

There is no way to “fix” this behavior with TCP

It is fundamentally how TCP works.

Therefore...



Tuesday, March 16, 2010

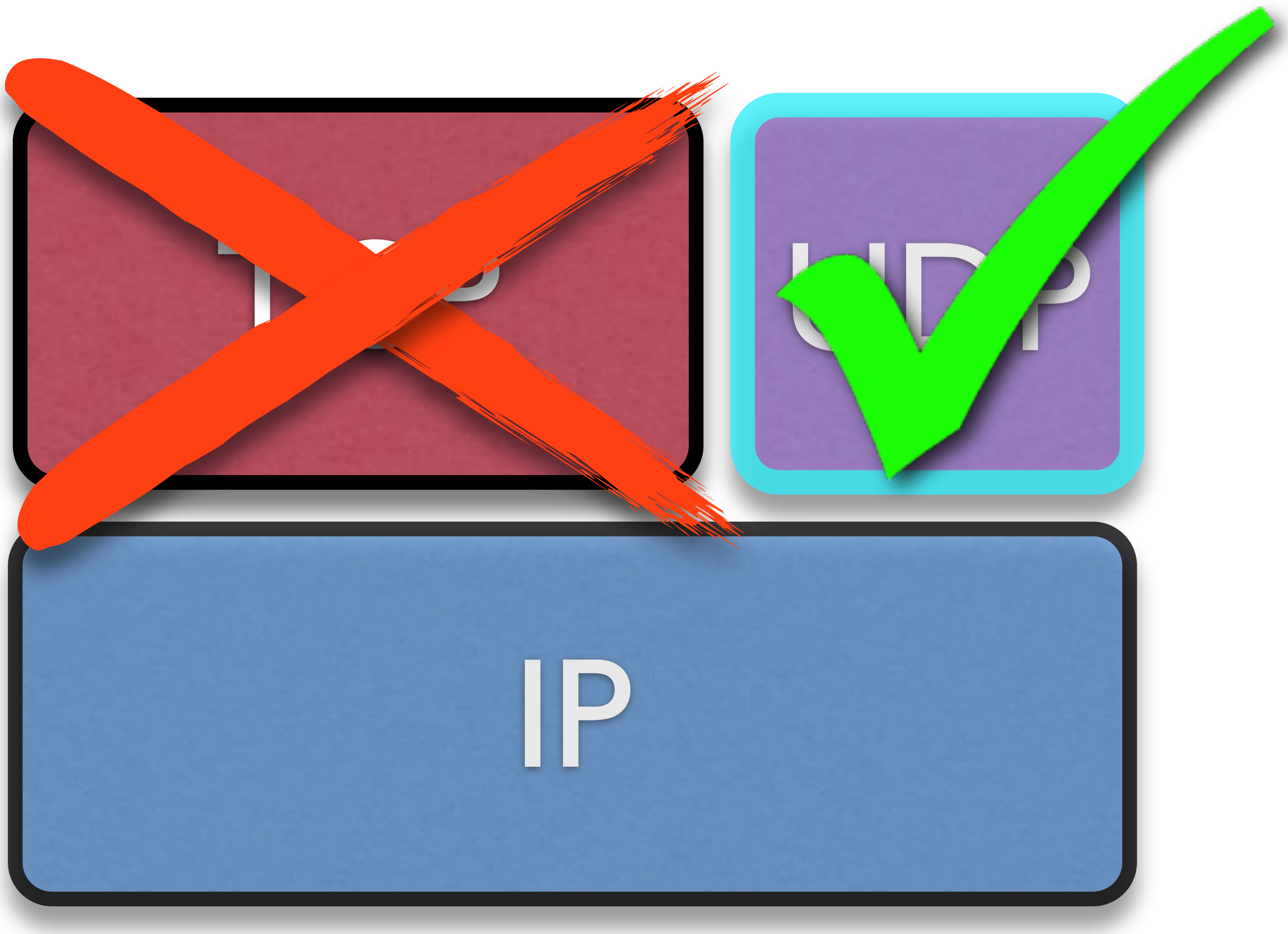
Use UDP instead

UDP works exactly like the underlying IP network.

Packets can be lost, received out of order.

If one packet is lost, you can process the next one without waiting for the lost packet to be resent.

Think about it – if information is time critical – what good is a resent packet anyway?



The most important thing to remember:

If you require real-time delivery of packets with minimum latency and jitter - **always use UDP!**

If all of your data is most-recent state, then you can use UDP add a sequence number at the top of your packet and everything will be fine. Just ignore out of order or lost packets and carry on.

If you need to deliver some data reliably, you can develop your own reliability system inside your UDP protocol

This is not really that hard to do and is a worthwhile exercise for any network programmer

See here for more info http://www.gafferongames.com/networking-for-game-programmers/reliability-and-flow-control

Alternatively, most networking middleware uses UDP and will do this for you - Raknet, OpenTNL, ENet etc...

DEMO

The demo shows two simulations side by side with one second simulated latency (for effect)

To see this for yourself, run the demo and press ALT-2 (note: MacOSX 64bit version only)

You should see two views of the world in splitscreen, the view of the left is the master which you control

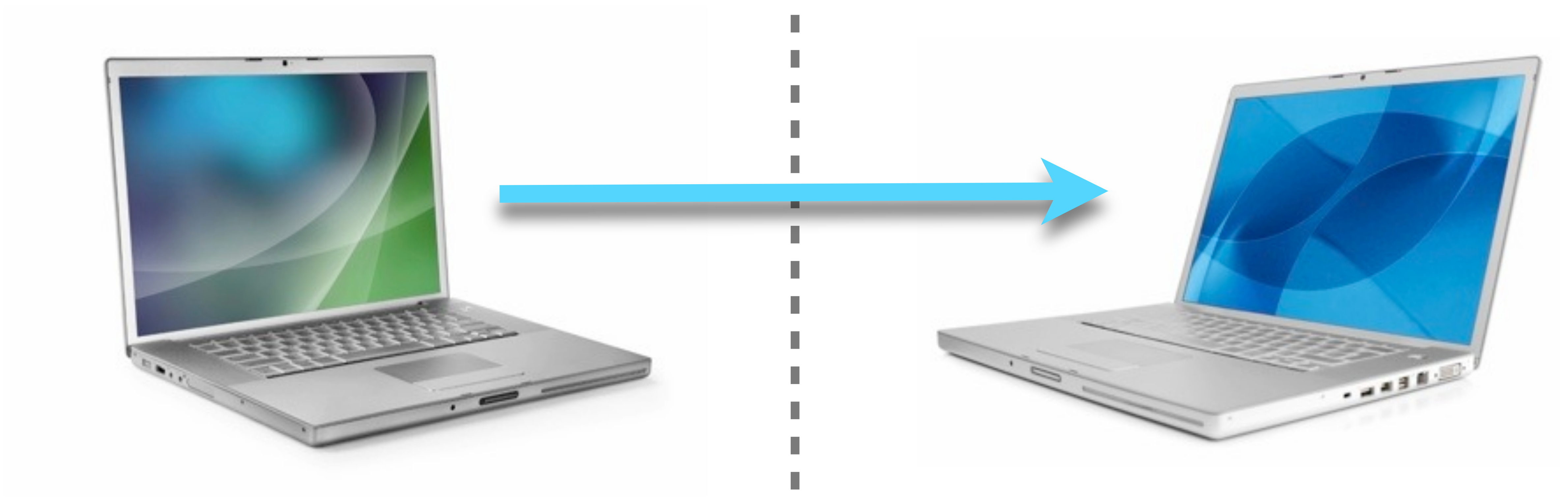
the view on the right is the slave, which has one second of simulated latency

what you do on the left can be viewed on the right... move around with arrow keys, press space to jump, hold space to hover. hold z to do katamari effect

Note that everything stays in sync!

Authority

Remote View



quantize physics state
pull input and state
serialize packet (write)
send packet

update(dt)

quantize physics state
receive packets
serialize packets (read)
push most recent input+state

update(dt)

how does it work?

most importantly: **we are running the simulation on both sides**

overview of main loop: authority on left is controlling the simulation

remote view just “sees” what happens on authority side -- no player controlling stuff on remote view, just one player here

first we quantize the state using whatever compression scheme we transmit with (if lossy) to ensure the same loss occurs on both sides - this improves determinism

next, note that when receiving packets we will not receive nice packets @ 30pps as were sent (see RTT graphs previously...)

some frames we will receive no packets, other frames we will receive clumps of packets

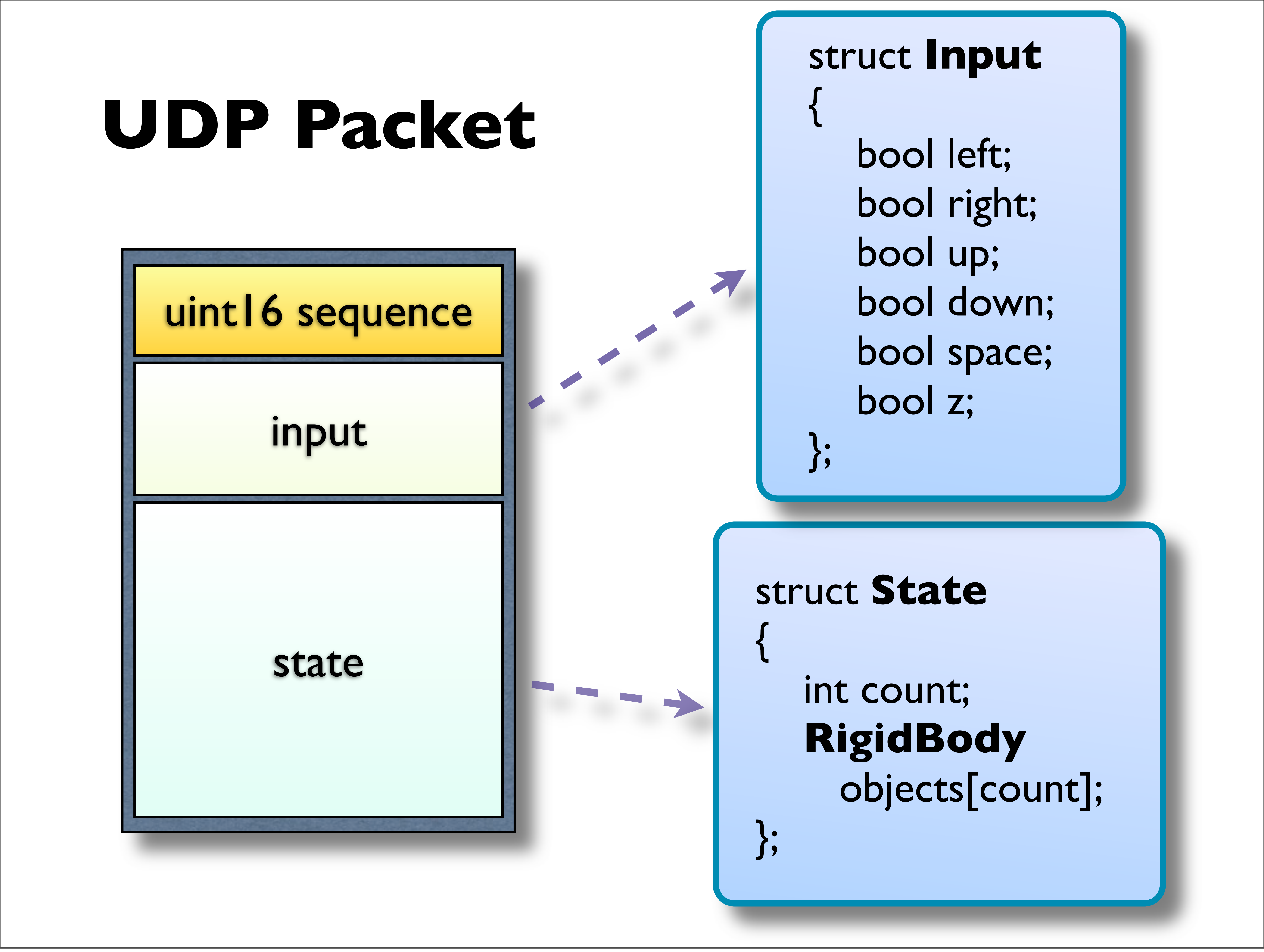
the simplest way to handle this is to include a sequence number in each packet. this number increases with each packet sent. on the receiver side at the beginning of each frame simply receive all queued up packets, if any -- then push only the most recent sequence number to the simulation.

In effect we are saying “we only care about the most recent state”.

Note that when we apply the physics state in the remote view we snap it hard. This works well for ODE but your mileage may vary depending on the physics engine you are using. Ideally, if you set the physics object in the remote view to exactly match the network state then they extrapolation should match very closely as well, at least for a second or so. In general you get a better extrapolation from a known good physics state than from attempting to apply a physics force to “push” an object towards where it should be (lame).

To cover up cases where you have discrepancies between the remote view and the latest state coming in over the network, I like to use a simple smoothing algorithm. I maintain an error offset and quaternion delta between the current state an the desired state coming in from the network. I snap the desired physical state hard, but track error offset which is used to visually smooth this error out over time. I use an exponentially smoothed moving average to reduce this error to zero over time, eg. position_error *= smoothing_tightness. This is done in local space so the smoothed view of the object does not lag at speed.

See the source code for more details.



UDP Packet

uint16 sequence

input

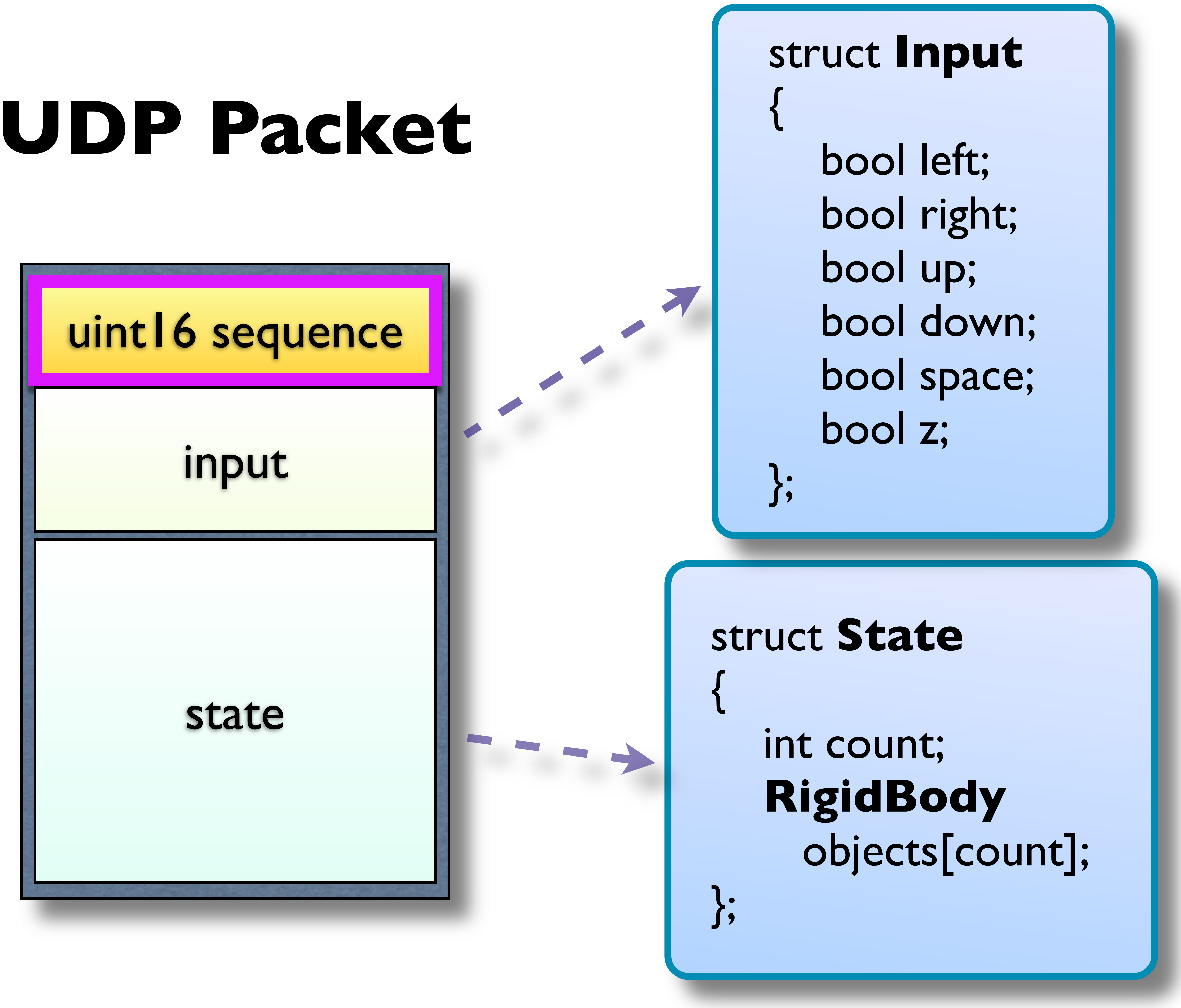
state

```
struct Input
{
    bool left;
    bool right;
    bool up;
    bool down;
    bool space;
    bool z;
};
```

```
struct State
{
    int count;
    RigidBody
        objects[count];
};
```

Here is the structure of the UDP packet we send.

UDP Packet



```
struct Input
{
    bool left;
    bool right;
    bool up;
    bool down;
    bool space;
    bool z;
};
```

```
struct State
{
    int count;
    RigidBody
      objects[count];
};
```

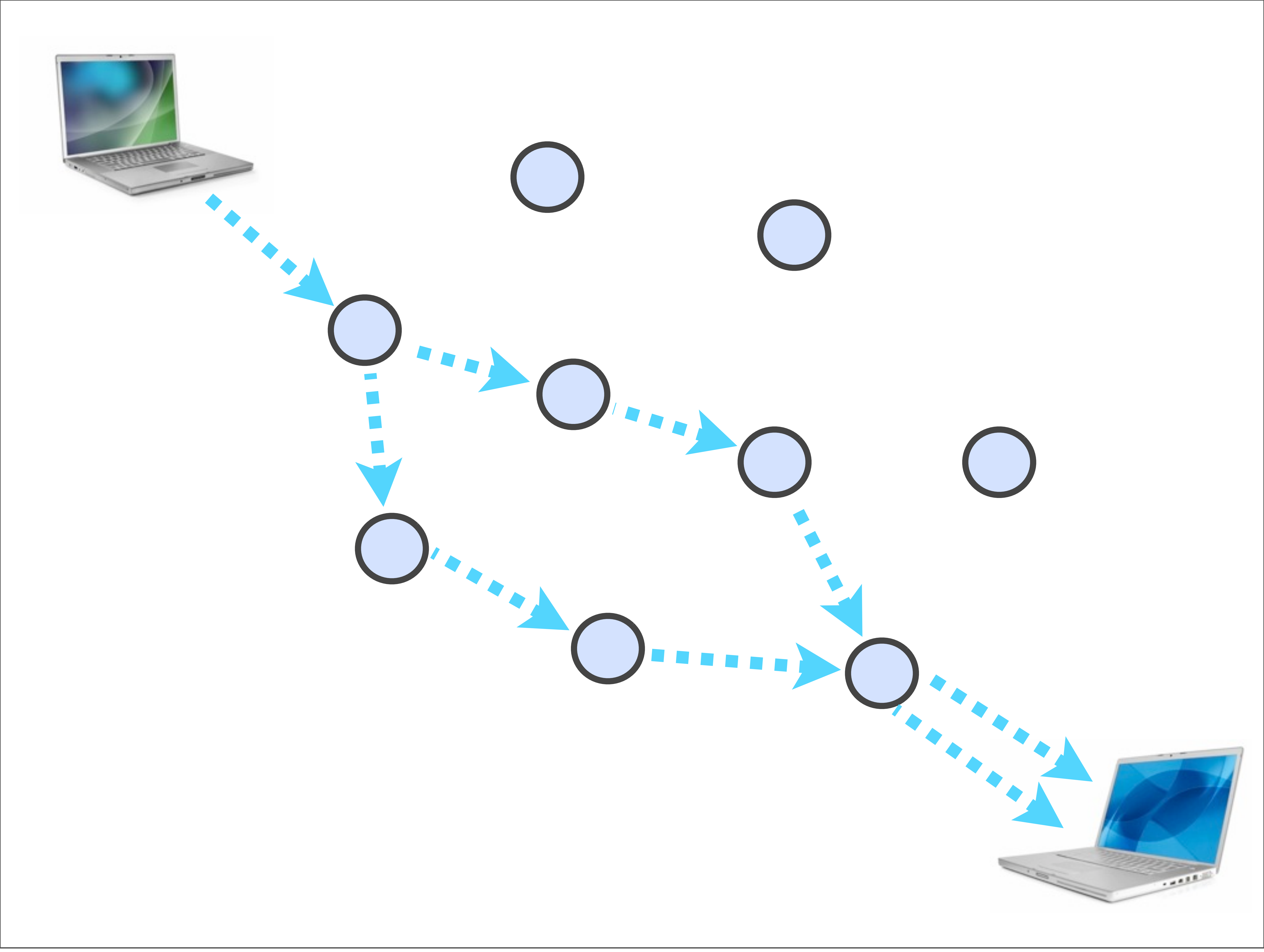
Tuesday, March 16, 2010

We include a sequence number at the top, the number is increased with each packed sent.

We only need 16 bits or so assuming 30pps because it is easy to handle wrap around.

Eg:

```
inline bool sequence_more_recent( unsigned int s1, unsigned int s2, unsigned int max_sequence )
{
    // returns true if sequence s1 is more recent than s2
    return ( s1 > s2 ) && ( s1 - s2 <= max_sequence/2 ) ||
           ( s2 > s1 ) && ( s2 - s1 > max_sequence/2  );
}
```



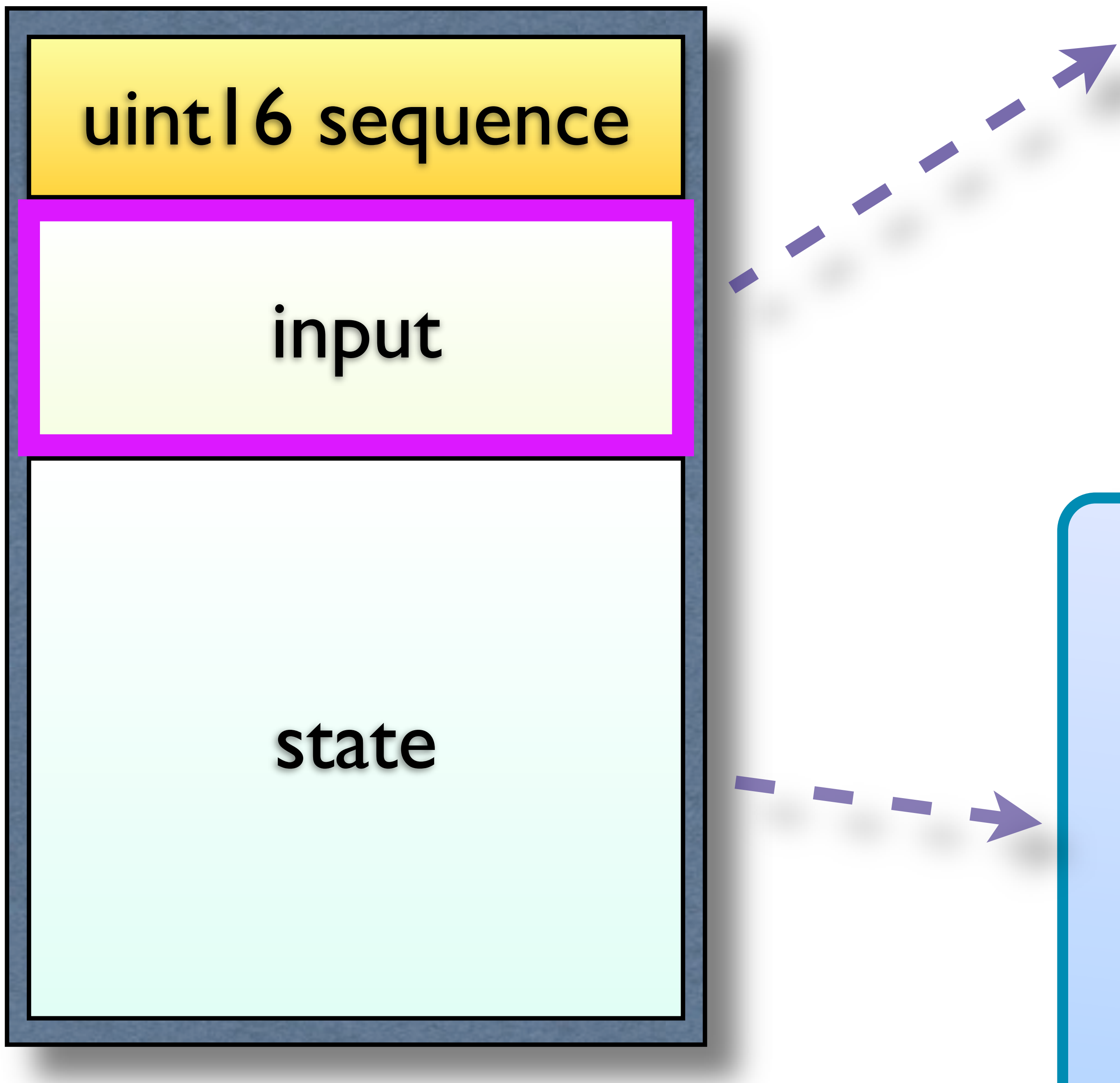
Tuesday, March 16, 2010

We need the sequence number because packets may arrive out of order, or in duplicate -- sometimes packets arrive really really late. sequence number lets us ignore these packets.

In summary, the sequence number is used to:

a) discard out of order packets
b) detect lost packets
c) push the most recent state only each frame, in the case when multiple packets are received per-frame

UDP Packet



```
struct Input
{
    bool left;
    bool right;
    bool up;
    bool down;
    bool space;
    bool z;
};
```

```
struct State
{
    int count;
    RigidBody
      objects[count];
};
```

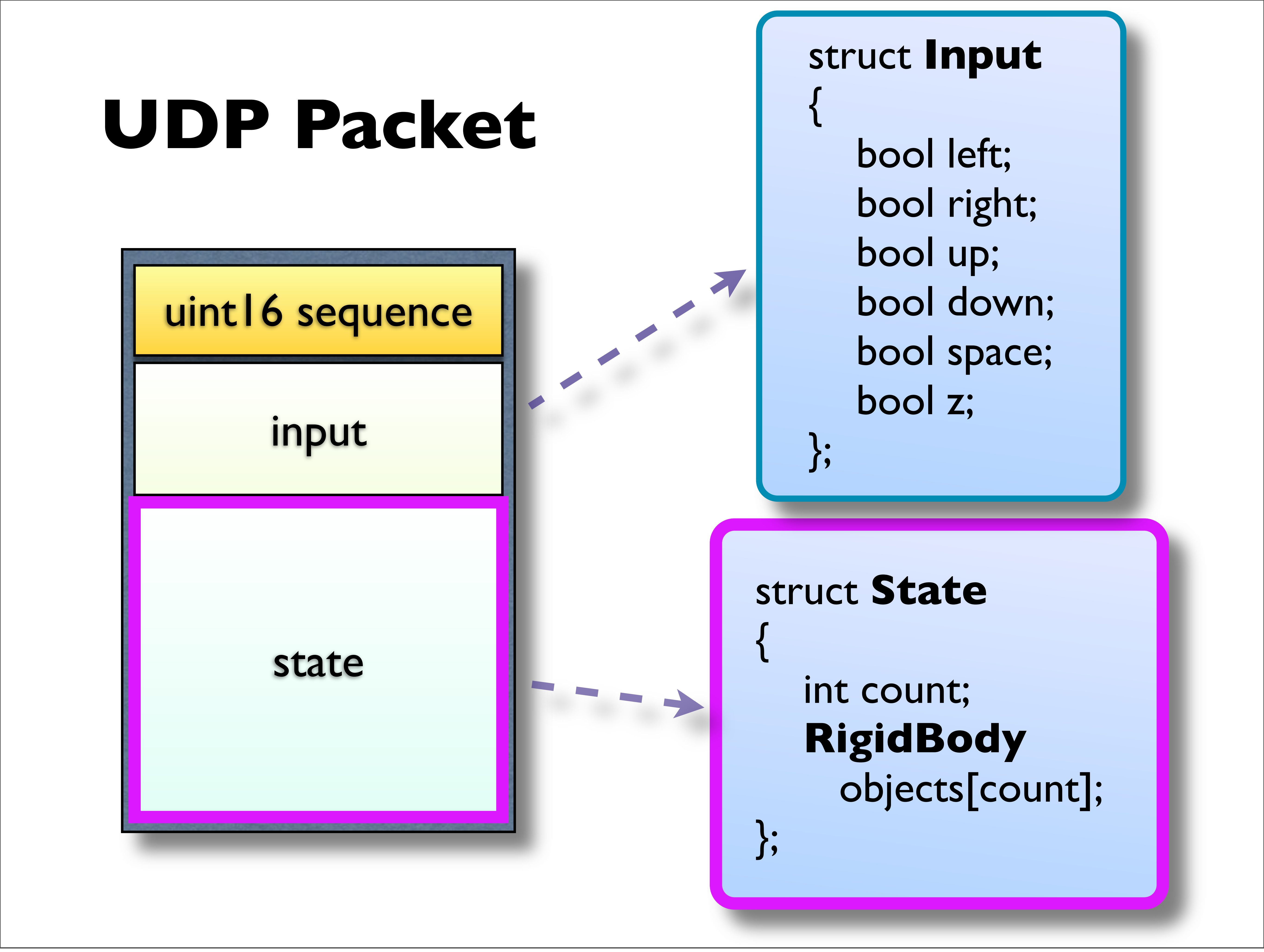
The player’s input is sent.

In this case left, right, up, down, space (and Z actually, for the katamari...)

If the remote view does not receive a packet in a given frame, it holds the same input as it last received

I’ve found that this generally gives the best extrapolation under packet loss, late packets for most cases in this specific demo

YMMV



UDP Packet

uint16 sequence

input

state

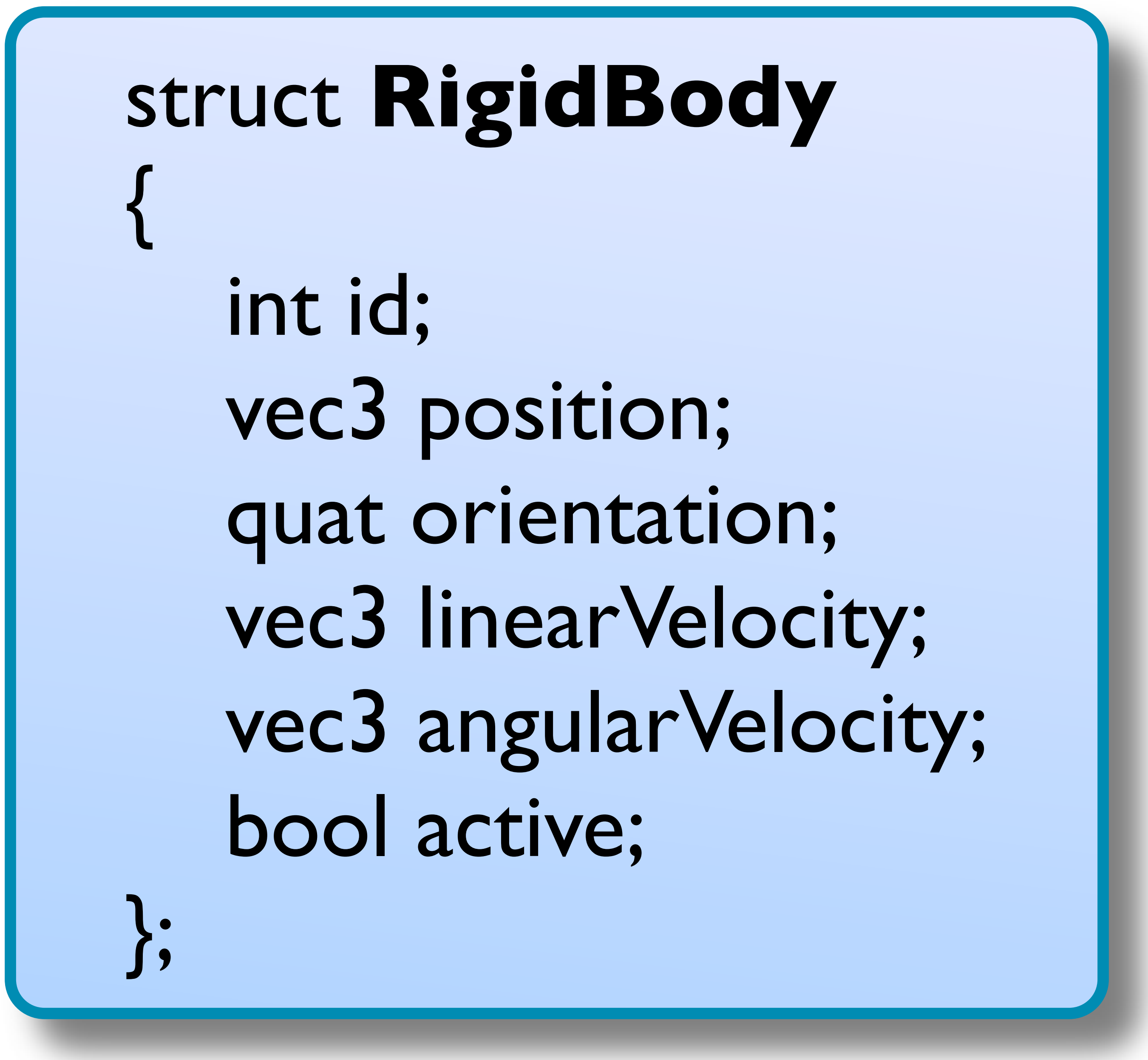
```
struct Input
{
    bool left;
    bool right;
    bool up;
    bool down;
    bool space;
    bool z;
};
```

```
struct State
{
    int count;
    RigidBody
       objects[count];
};
```

Tuesday, March 16, 2010

State is a count = n, followed by n rigid bodies.

I cram as many rigid bodies into the packet as I can.



```
struct RigidBody
{
    int id;
    vec3 position;
    quat orientation;
    vec3 linearVelocity;
    vec3 angularVelocity;
    bool active;
};
```

Each cube has the following rigid body state:

id,
position
orientation
angular velocity
linear velocity
active (true if active, false if at rest)

that’s 12 bytes for each vector, 16 bytes for the quaternion, and another 4 bytes so we can identify the cube over the network

That’s 57 bytes per-rigid body

!!!

You can compress this:

id is in range [1,1000000] or so, does not need full 32 bits, bitpack it

position xz can be compressed relative to origin point of circle, send the origin once at the beginning of the packet quantized to 1meter res (int) -- then encode a +/- 10m or so @ 1mm precision as integer values

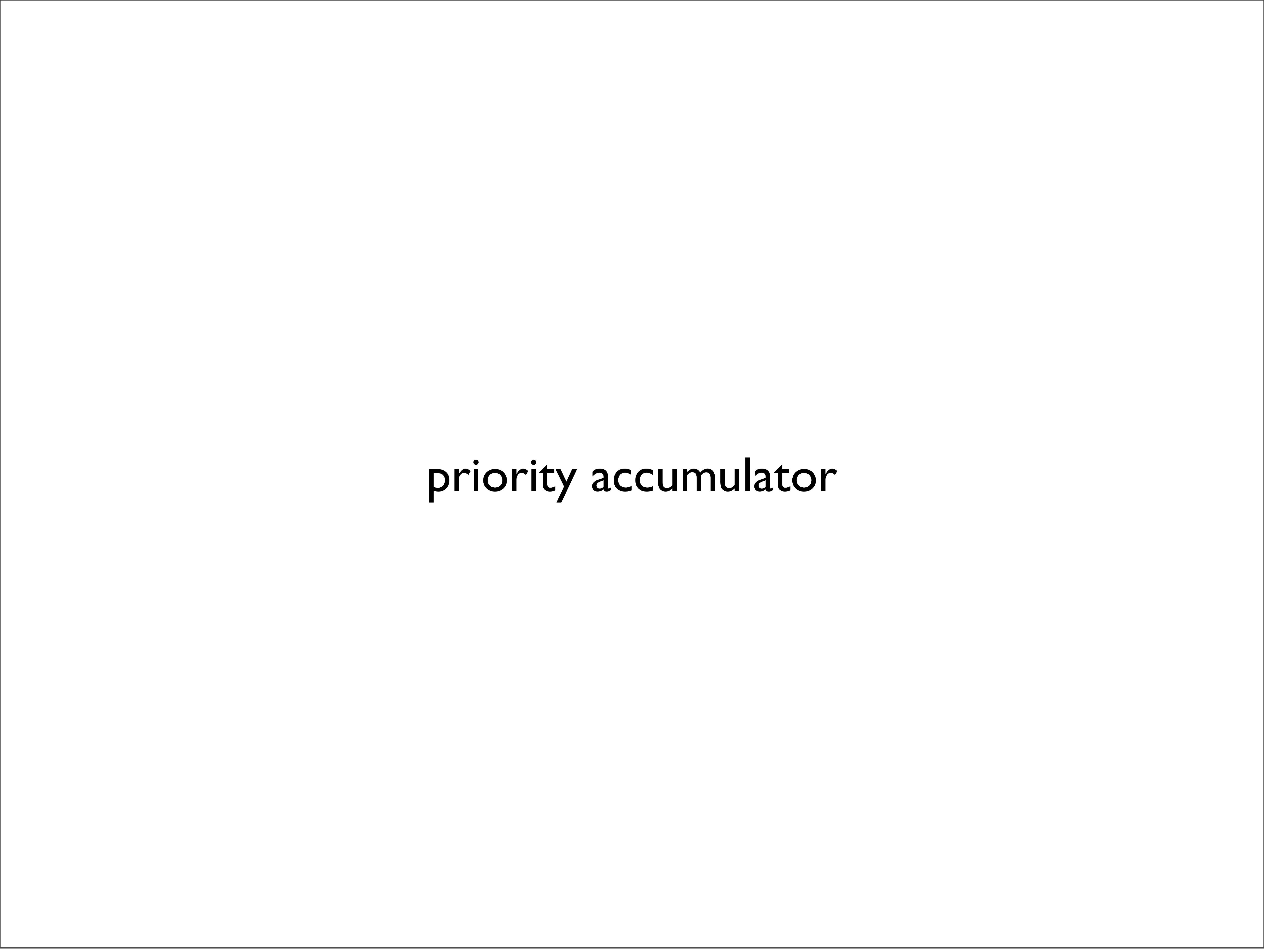
position y encoded by constraining object in range [0,max_height] and quantizing to 1mm or so

quaternion compressed to 32bits using smallest 3 method (see game gems article). very easy to compress, can get away with less than 32 bits even.

lin/ang velocity compressed by limiting max velocity and compressing in range -- see source code.

important: active means “not at rest”. if active = false, no need to send lin/ang vel over network, assumed zero. saves a lot of bandwidth

also note - when you do this compression, make sure you quantize all objects on the sender side at the beginning of each frame, as you would when packing them into a packet, this ensures that the simulation in the remote view extrapolates from the same initial state that the authority side starts with)



priority accumulator

Tuesday, March 16, 2010

Even with compression, say 20 bytes per-rigid body instead of 70+ -- you can only fit about 10 or so objects into a 220 byte packet (assuming header etc. @ 64kbit/sec)

Clearly we have more than 10 active objects -- how to distribute updates across the scene, while prioritizing some objects over others?

The solution is simple: priority accumulator

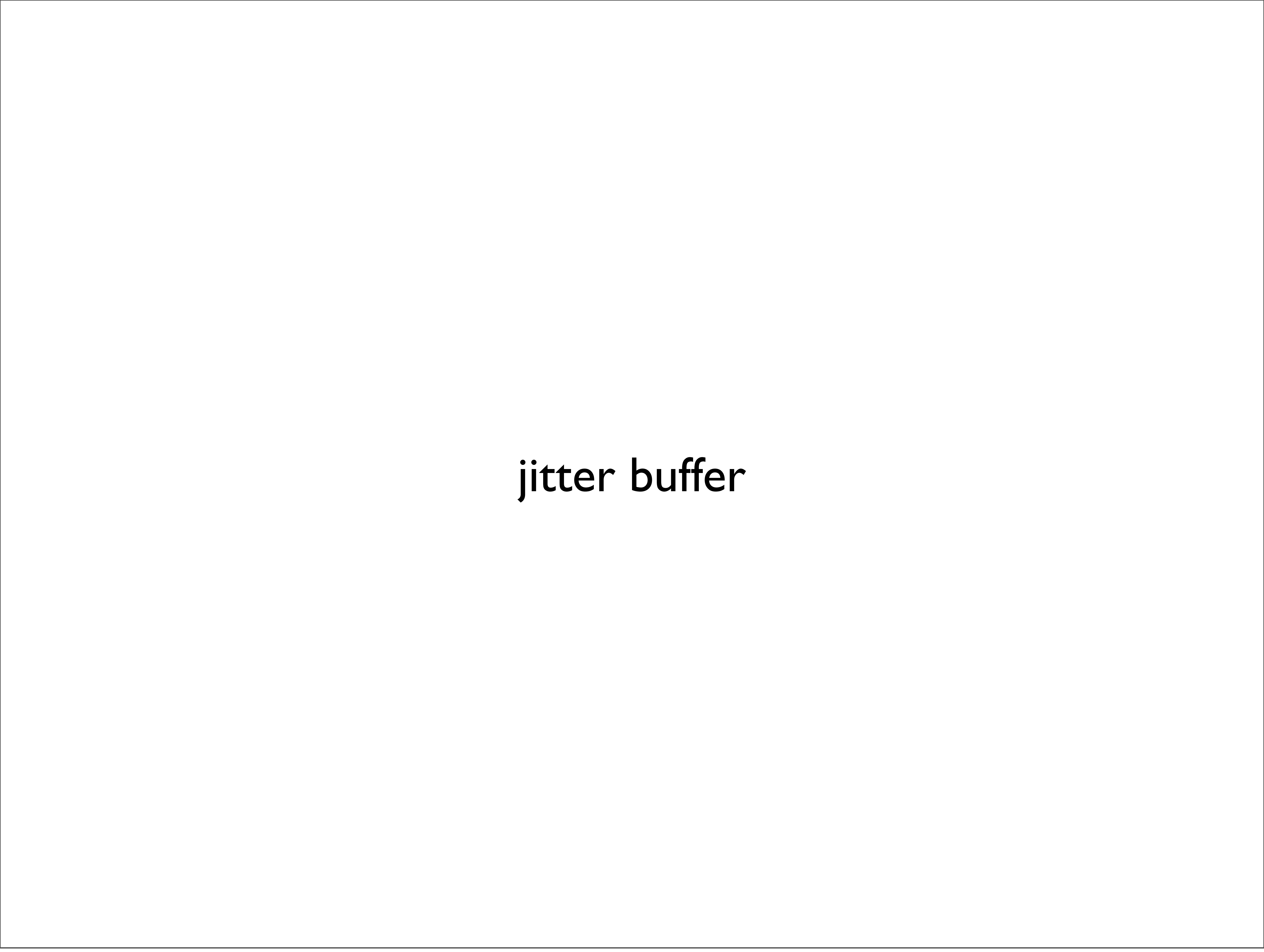
Each object has priority accum float value -- each frame you calculate dt * priority scale for each object.

Each packet sent... grab n highest priority objects. serialize as many as will fit into the packet. Each object which goes into sent packet has priority set back to zero.

Next frame, priority accumulates scaled by dt again. high priority objects bubble up quickly, while low-priority objects still get included in packets occasionally.

Therefore updates are distributed fairly, while still prioritizing some objects over others.

In this demo, i prioritize objects i interact with (same color as my cube) over stationary objects -- this works pretty well. if the circle was much larger, i'd prioritize by distance as well, and maybe view frustum and so on... you get the general idea.



jitter buffer

Tuesday, March 16, 2010

one final thing before we move on...

instead of just pushing most recent state you can consider using a jitter buffer

this means that instead of pushing most recent state, you hold packets in jitter buffer for 100ms before pushing to sim

effectively, this trades latency and packet loss (late packets = last packets) for smooth delivery of packets spaced evenly @ the time they were sent, eg. 30pps.

jitter buffers are used a lot in VoIP, it seems easy but it is actually reasonably complicated (!)

google for “jitter buffer” or “playout delay buffer” --- for more advanced stuff, look for papers on “adaptive jitter buffer”, “adaptive playout delay”

but here is my quick summary, include timestamp in packets, try to speed up / slow down play out to adjust to keep X ahead of average (or min) timestamp coming in from authority in remote view. trickier than it seems, given IP “best effort” packet delivery -- requires a lot of testing and tuning to get right, especially when the baseline RTT may fluctuate over the course of a connection.

DEMO

Demo showing 4 players

To see this run the demo then press ALT-4 to start the cube demo (ALT-5 if you are in the Win32 MIGS'09 demo)

(You should see random sized red cubes raining down, plus one big red cube controlled by the player)

Then press 6 to view top-down

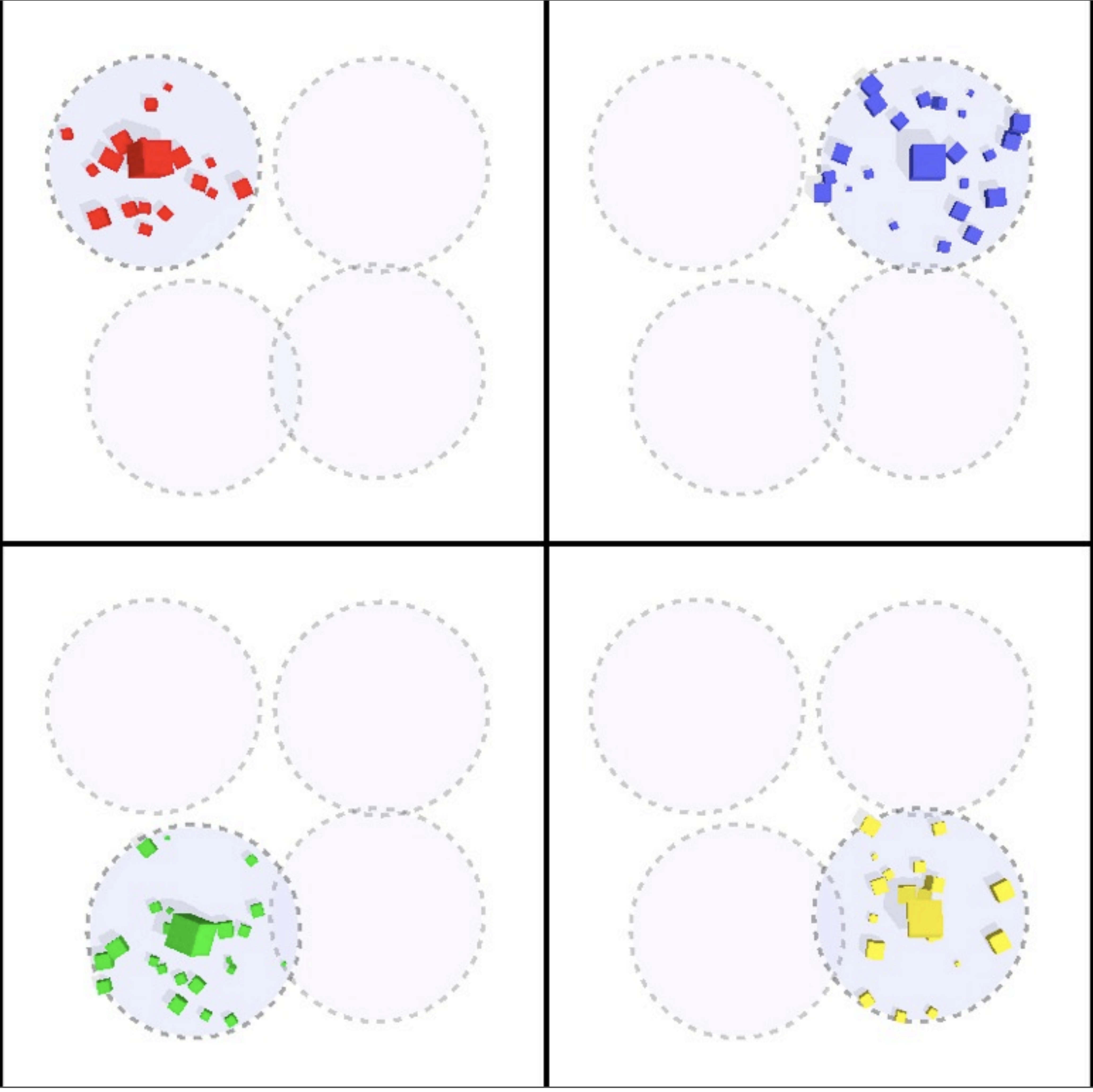
Press tab once to view the other player's activation circles

Press enter once to see splitscreen, then enter again to view quadscreen

Now you see each player has 4 separate worlds of 1 million cubes each

Note that each player only activates objects inside their own activation circle

There is currently no communication between each player's world (verify by moving about using arrow keys, press 1,2,3,4 to select current player and arrow keys to move around)



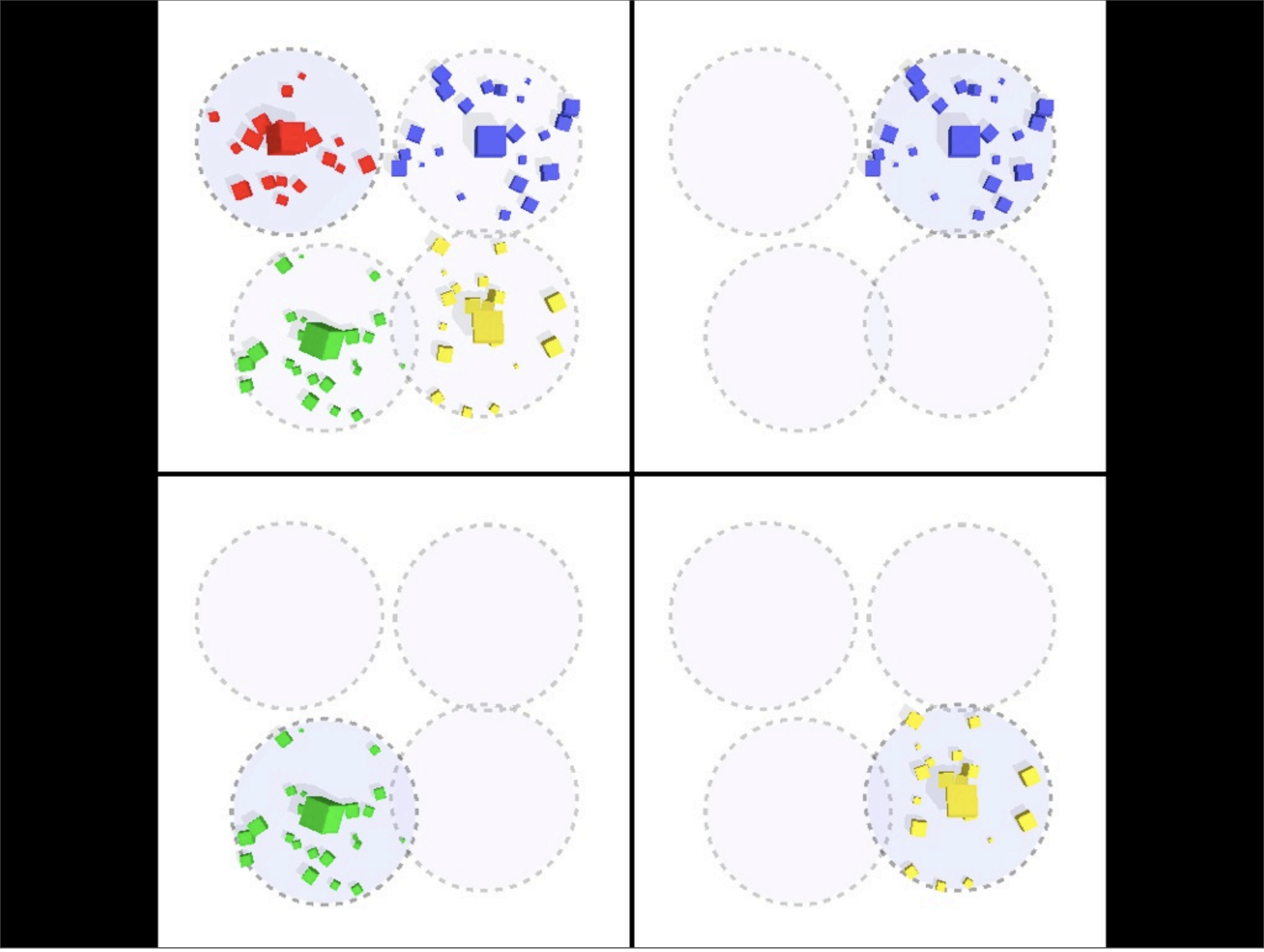
Tuesday, March 16, 2010

We start with four separate worlds each containing one million cubes.

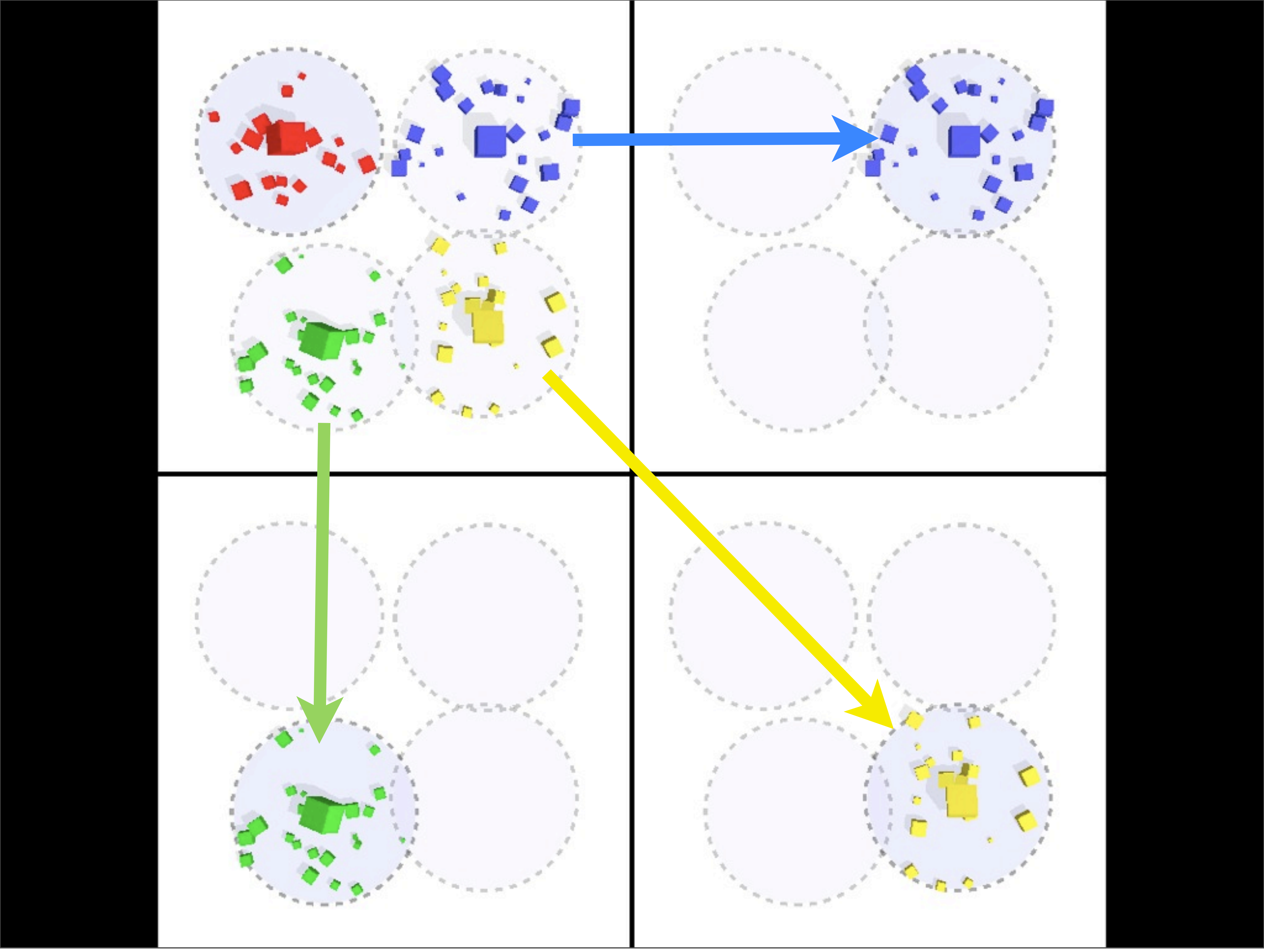
Normally, each player only activates cubes inside their circle - lets assume that in this arrangement they are already CPU bound

There is no synchronization between the worlds at this point...

How can we keep all four simulations in sync?



One option would be to designate one of the players to be the server in a typical client/server setup.



The server player becomes the “authority side” for all objects.

Each frame the clients send their input to the server, and each frame server sends out the the state of where objects are in the world.

However, there are some problems with this...

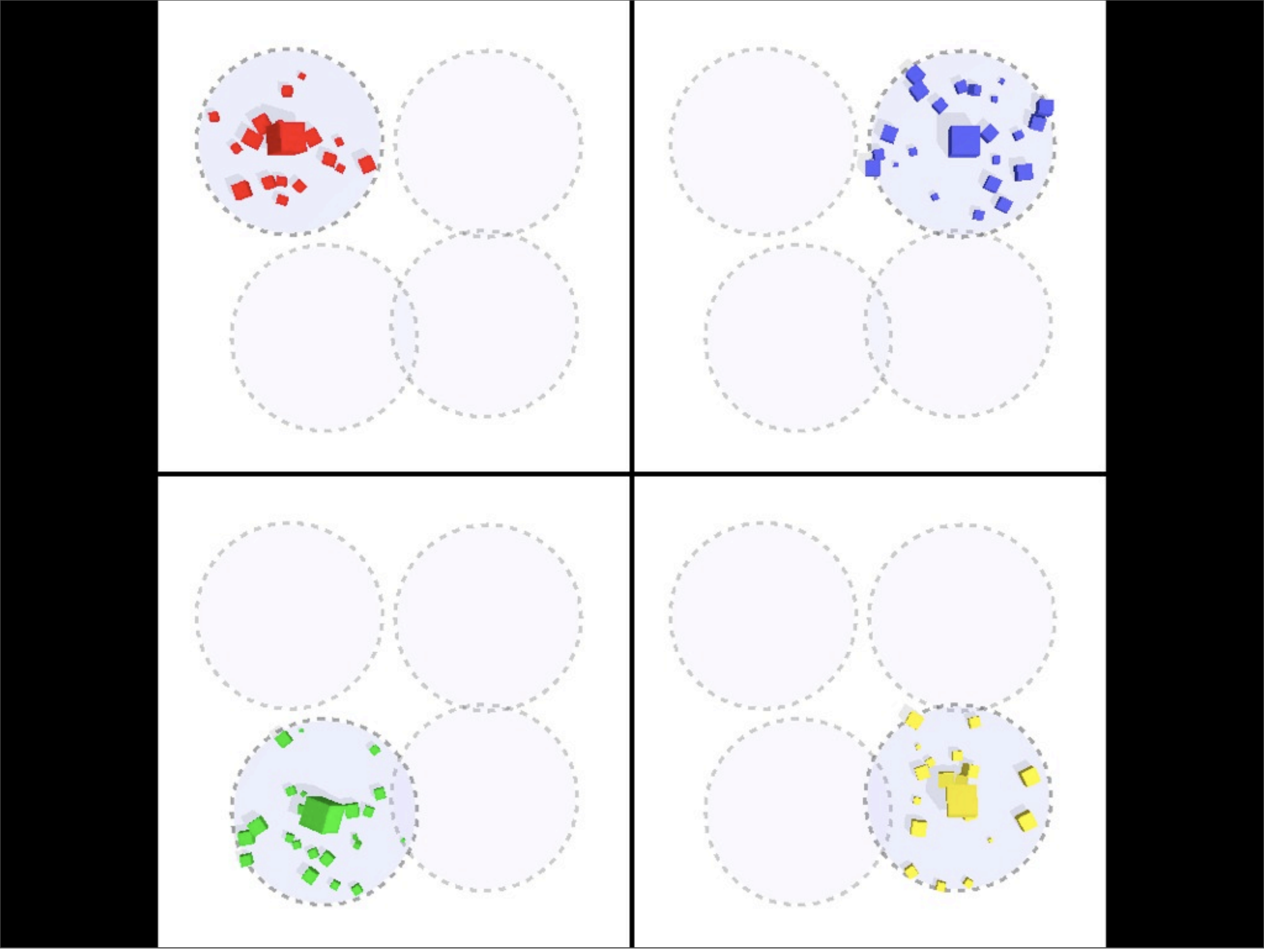
Firstly, clients feel latency because they do not see the simulation respond to their input until the state comes back from the server

Secondly, the server player has 4X the CPU load that he does in singleplayer (note he has four circles)

This is a serious issue because physics simulation is generally very expensive, I’d hate to have to reduce the game world to only 1/4 of the CPU budget just to support multiplayer.

One solution for problem this would be to host a dedicated server, this way the server could be very powerful and able to simulate 4X the objects. but it would cost a lot of money to provide one dedicated server for every four people who want to play the game...

Because of this I started to look into a peer-to-peer technique for synchronizing this game world instead.

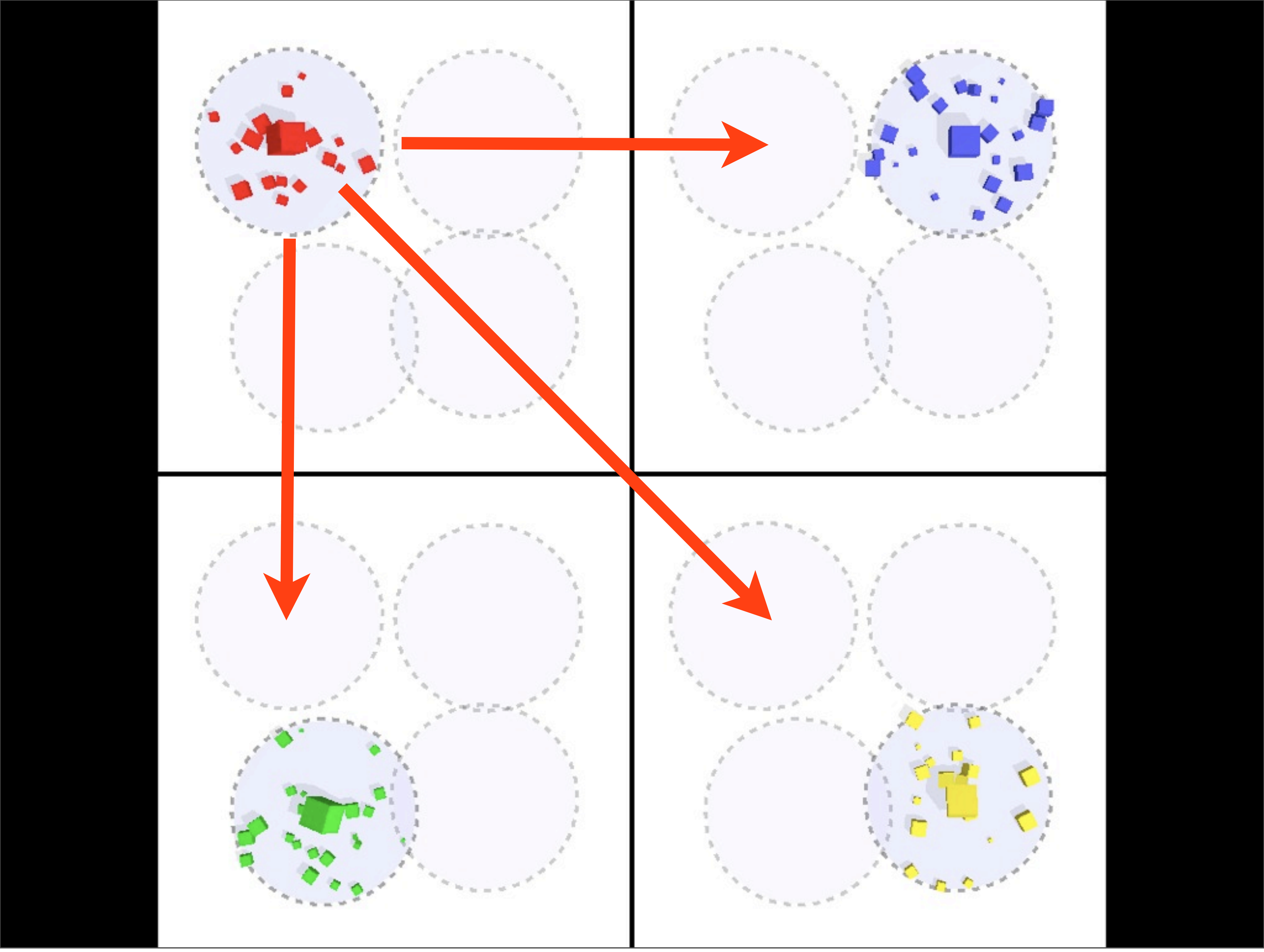


Tuesday, March 16, 2010

What we want is for each player to only activate the objects which are around them, avoiding extra CPU, memory or streaming cost in multiplayer compared with singleplayer.

Because this is a COOP game we are willing to play a bit fast and loose and not be concerned with cheating

(If you are a competitive game then I highly recommend you look into deterministic lockstep networking, or client/server techniques instead)

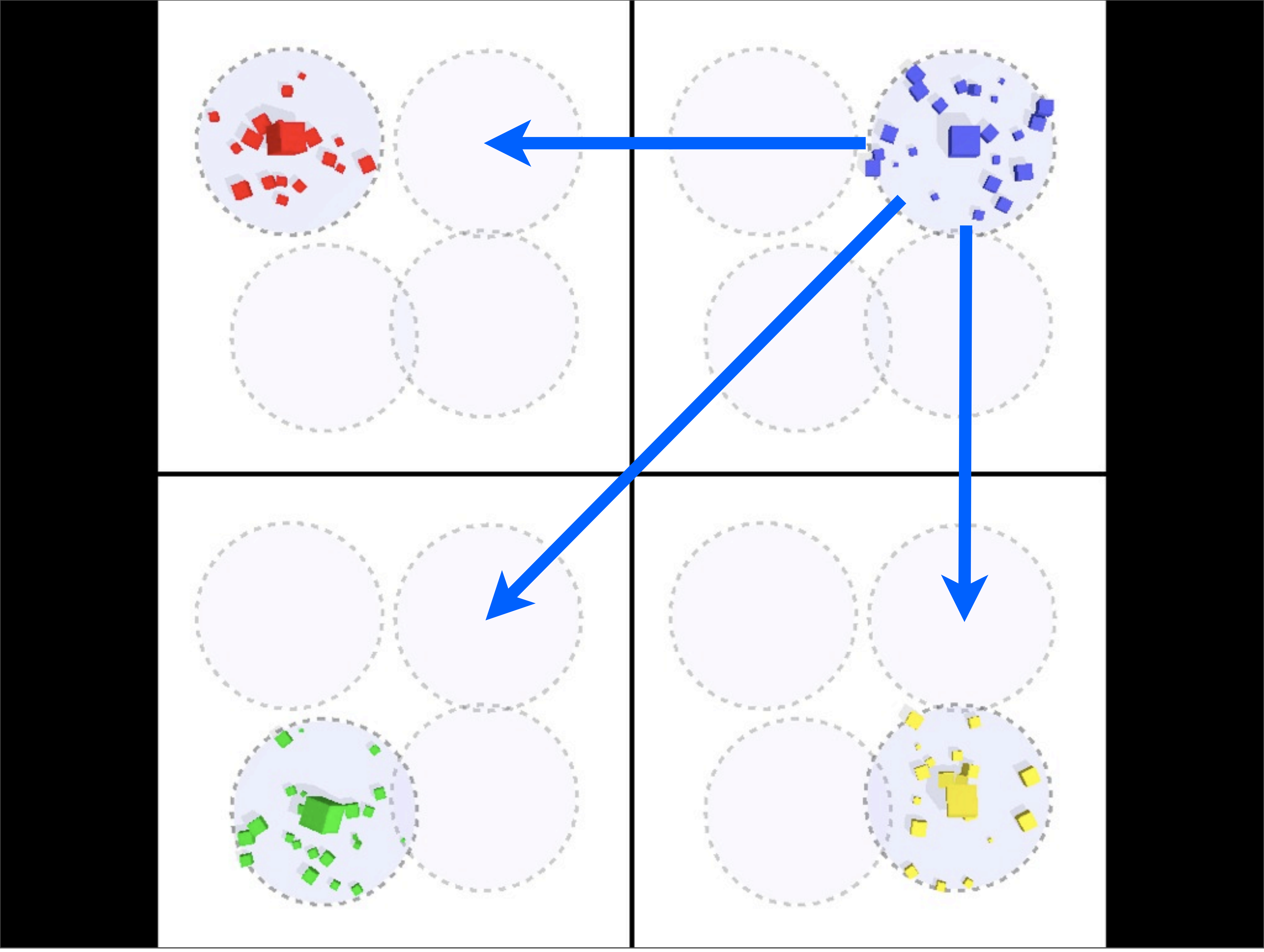


Tuesday, March 16, 2010

First lets try something naive...

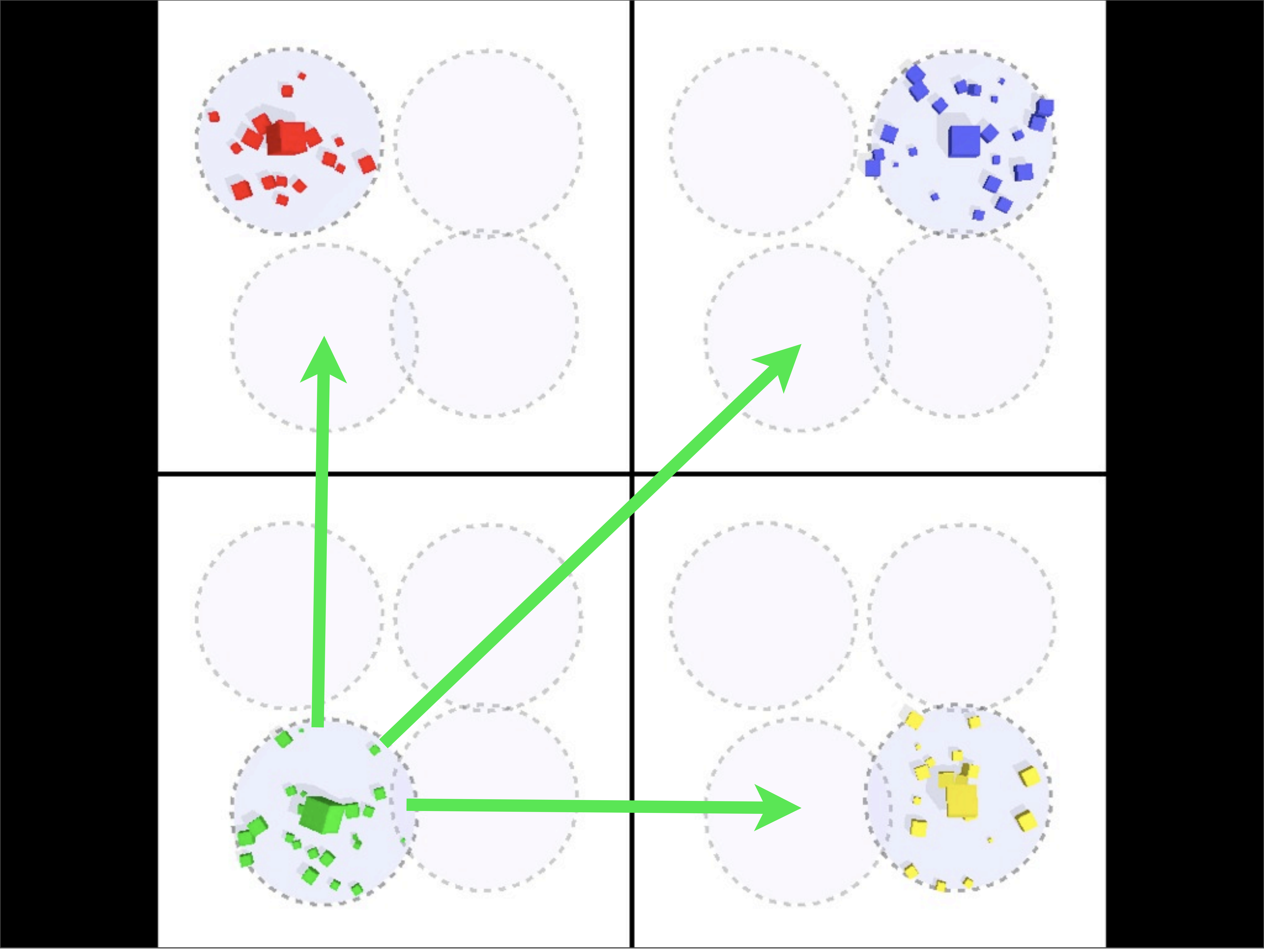
Each player just sends their input and the state of the objects inside their circle (their “active objects”) to all other players.

IMPORTANT: please remember that the entire world state is persistent, therefore the red player is able to set the state of the cubes inside his circle on the blue player’s machine -- even though those objects are <u>not active</u> on the blue player’s machine.



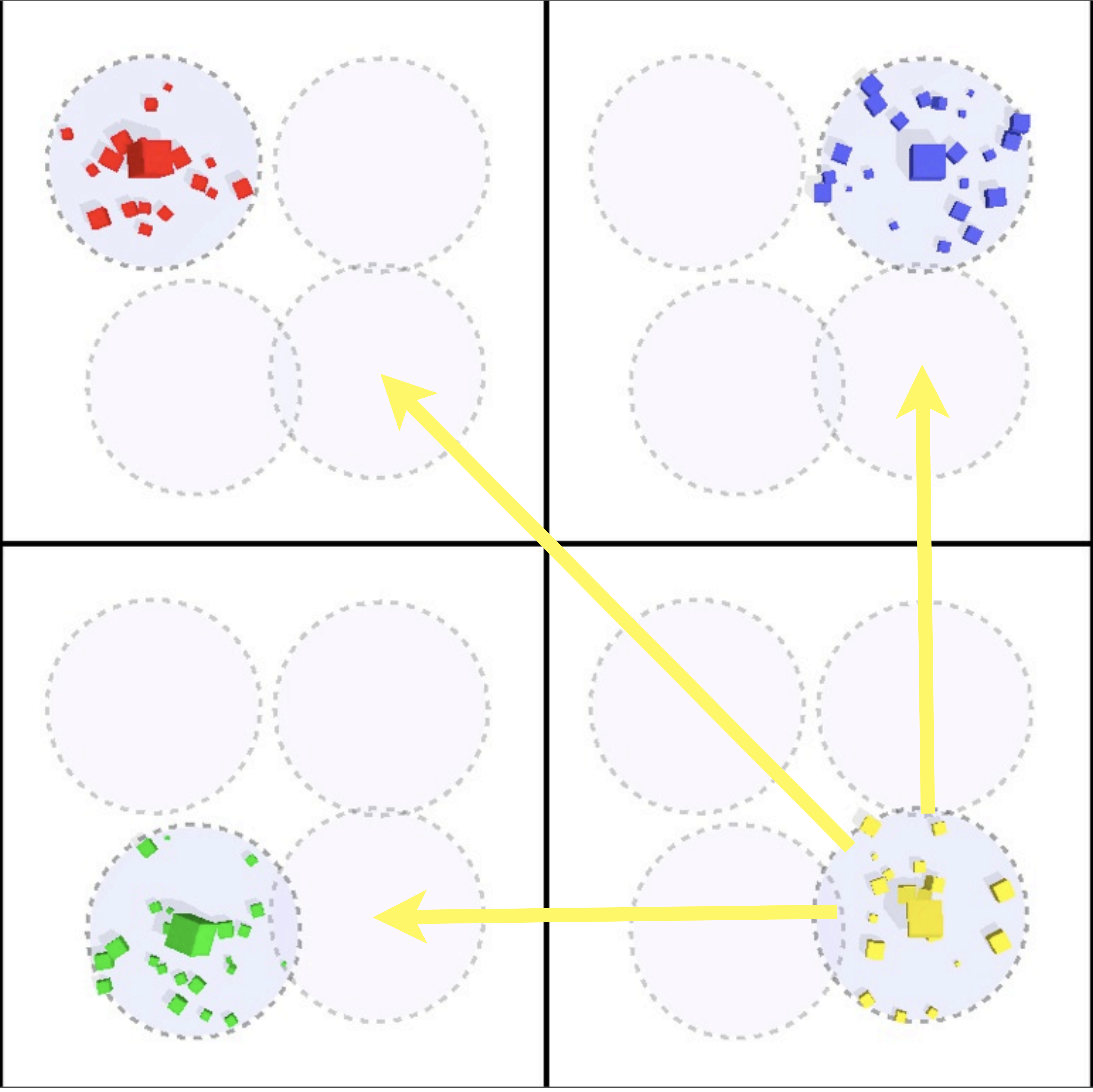
Tuesday, March 16, 2010

The blue player sends his active objects to all other players.



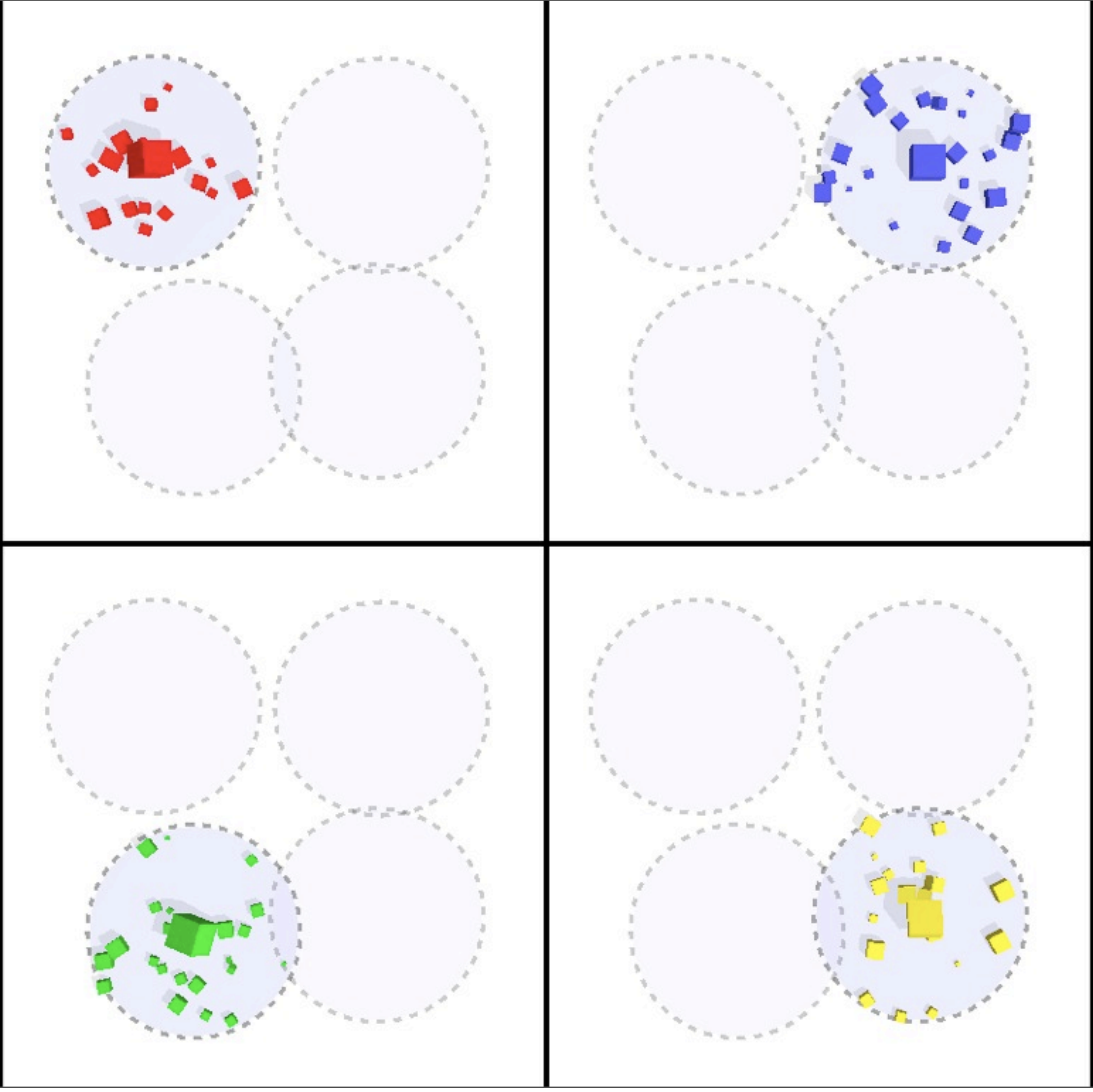
Tuesday, March 16, 2010

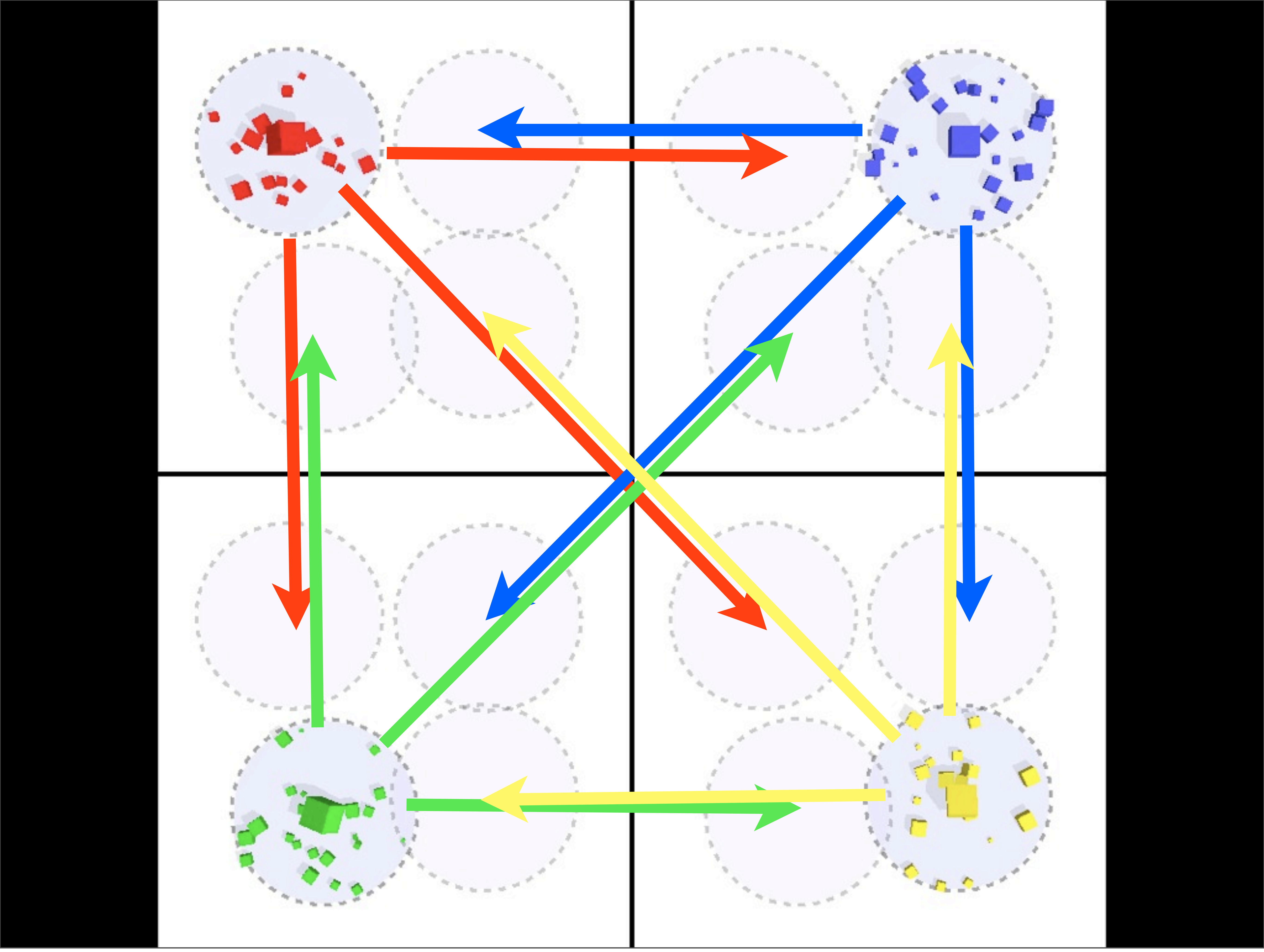
Green player does the same



Tuesday, March 16, 2010

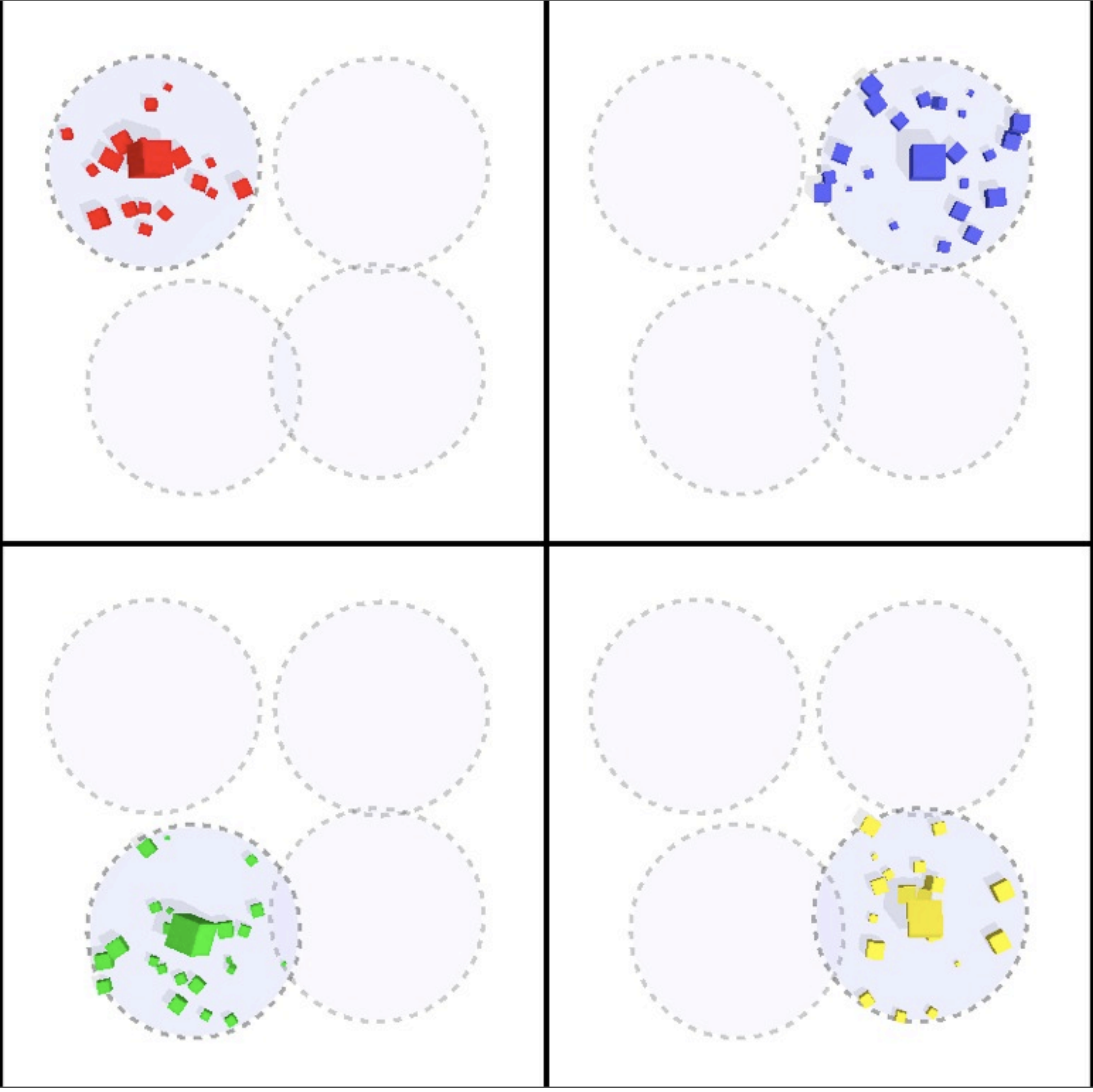
as does the yellow player





Tuesday, March 16, 2010

The total data flow looks like this – it’s complicated!



DEMO

Tuesday, March 16, 2010

Demo shows a few things...

First that there is no synchronization on by default. Each player has their own set of 1 million cubes, all set to the same initial state, but they quickly diverge to different places on each player's machine.

Next it shows that provided that the player's circles don't overlap, everything works fine -- objects stay in sync.

BUT

When the players circles overlap there is a massive jittering. This is feedback.

The problem is that there is no clearly defined direction of flow. In areas of overlap players fight over the position of objects.

What we need to do is work out some way to ensure that state flows only one direction -- from authority to non-authority.

But we don't have a server which can arbitrate where an object should be... just four peers, each with their own unique view of the world

How can we ensure that all four players agree on the state of an object in the world?

DEMO:

To reproduce this in the demo, press ALT-3 to select authority demo (or ALT-4 in MIGS'09 win32 build)

Then press tab three times and you'll see other player object cubes

Next press F2 to turn on sync (it's off by default) -- then move the red player cube into the blue player circle using the arrow keys

Roll the cube around a bit inside the blue player's circle and you'll very quickly see lots of jittering in movement

Move the cube outside the circle and the jitter disappears.

You can select other players via 1,2,3,4 then move them around too. Try overlapping all player circles for extra crazy!

Authority

Like *being the server* for an object

Tuesday, March 16, 2010

The solution in this case is to use an authority scheme.

Think of authority as like “being the server” for an object

Authority defines a direction of flow... state for an object flows from authority to non-authority. The non-authority side is the “remote view” of an object as described in the earlier slides synchronizing a simulation from left -> right.

In other words, if a player has authority over an object, they get to tell other players what the state of the object is.

This fixes the feedback effect when player circles overlap.

Authority Rules

1. Player Authority
2. Tie-Break Authority
3. Interaction Authority

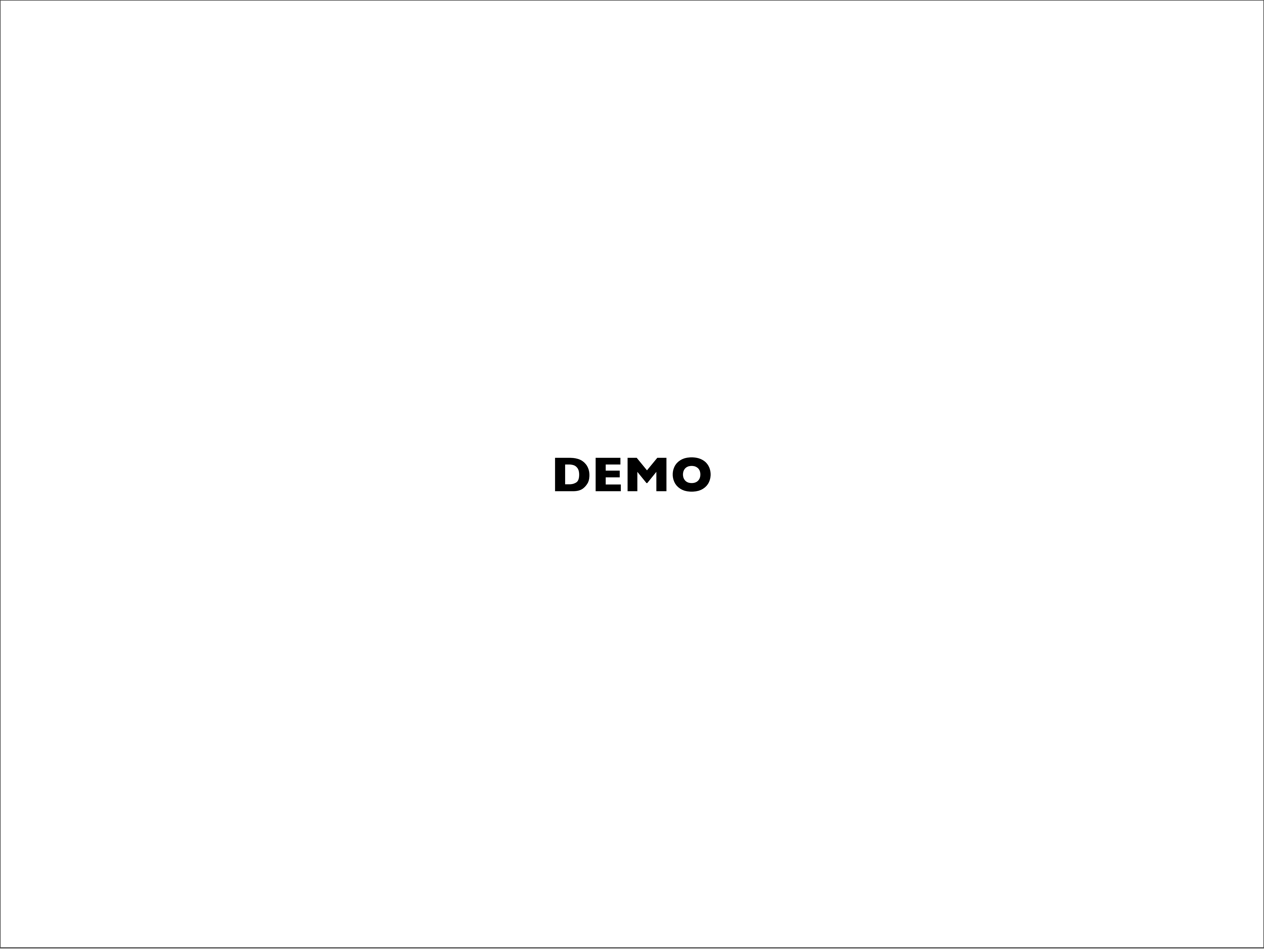
three rules solve all problems

player authority: players always have authority over the object they are controlling (e.g their cube)

tie-break authority: in areas of overlap, for grey cubes with default authority -- lowest player id wins. eg. in regions where red circle and blue circle overlap, the red player has authority and tells blue where grey cubes are.

interaction authority: players take authority over objects they interact with -- effectively becoming the server for that object, until it returns to rest and becomes grey again. this last technique hide latency for the higher player ids in areas of overlap.

IMPORTANT: these techniques also make the game extremely vulnerable to cheating, please use this technique for coop games only!



DEMO

Demo shows player authority -> default authority -> interaction authority

----

DEMO:

To reproduce this in the demo, press ALT-3 to select authority demo (or ALT-F4 in MIGS’09 win32 build)

Then press tab three times and you’ll see other player object cubes

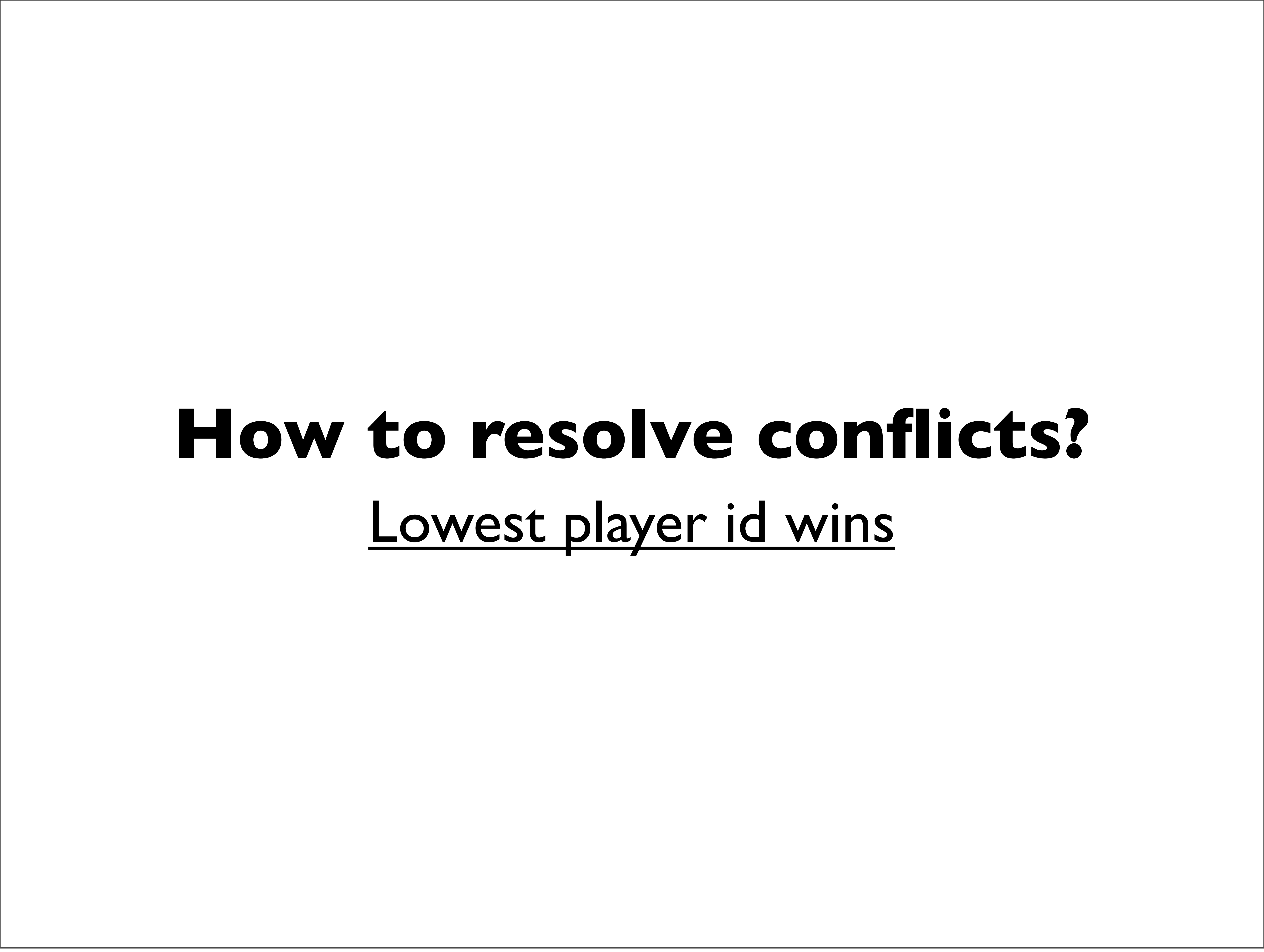
Next press F2 to turn on sync (it’s off by default) -- then move the red player cube into the blue player circle using the arrow keys

Roll the cube around a bit inside the blue player’s circle and you’ll very quickly see lots of jittering in movement

Press F3 to turn on player authority (jitter on player cubes goes away)

Press F4 to turn on default authority (jitter on grey cubes goes away)

Press F5 to turn on interaction authority (no jitter, and player feels no latency when interacting with other objects)



How to resolve conflicts?

Lowest player id wins

Players grab interaction authority locally without waiting for confirmation

It is possible that both players think they have authority over an object if there is a lot of lag (eg. 1 second) because they do not see the other players interaction with that object until after their own

How to resolve this conflict?

In this case the lowest player id wins

Lowest player keeps authority, higher player loses it and the object pops to correct state from higher player id (this pop is smoothed out visually...)

See the source code for implementation details.

Late Joins

Tuesday, March 16, 2010

Somebody joins in the middle of the game

We have 1 million cubes, i don’t want to stop the entire game and zip up state for 1 million cubes when somebody joins...

I want it to be seamless -- Ideally, i want to stream in just the fixes to world state as players need them

That’s what this section is all about!

Problem: Joining after world is changed

Tuesday, March 16, 2010

Red player changes the state of the world, then the blue player joins -- the blue player does not know about these changes

DEMO

In the demo the red player clears a swath, then the blue player joins -- blue player does not see it ... but instead sees regular grid of cubes, eg. initial state -- why?

DEMO:

To reproduce this in the demo, press ALT-4 to select corrections demo (or ALT-5 in MIGS'09 win32 build)

Then press tab once to see other player circles

Hold space then move the red player with arrow keys to clear a horizontal line, including right through the blue player's circle

Return the red player back to his initial position (press 6 for overhead view, line him up in top-left)

Press enter twice to go to quadscreen.

There is no sync on right now, note that the blue player does not see the line cleared by red

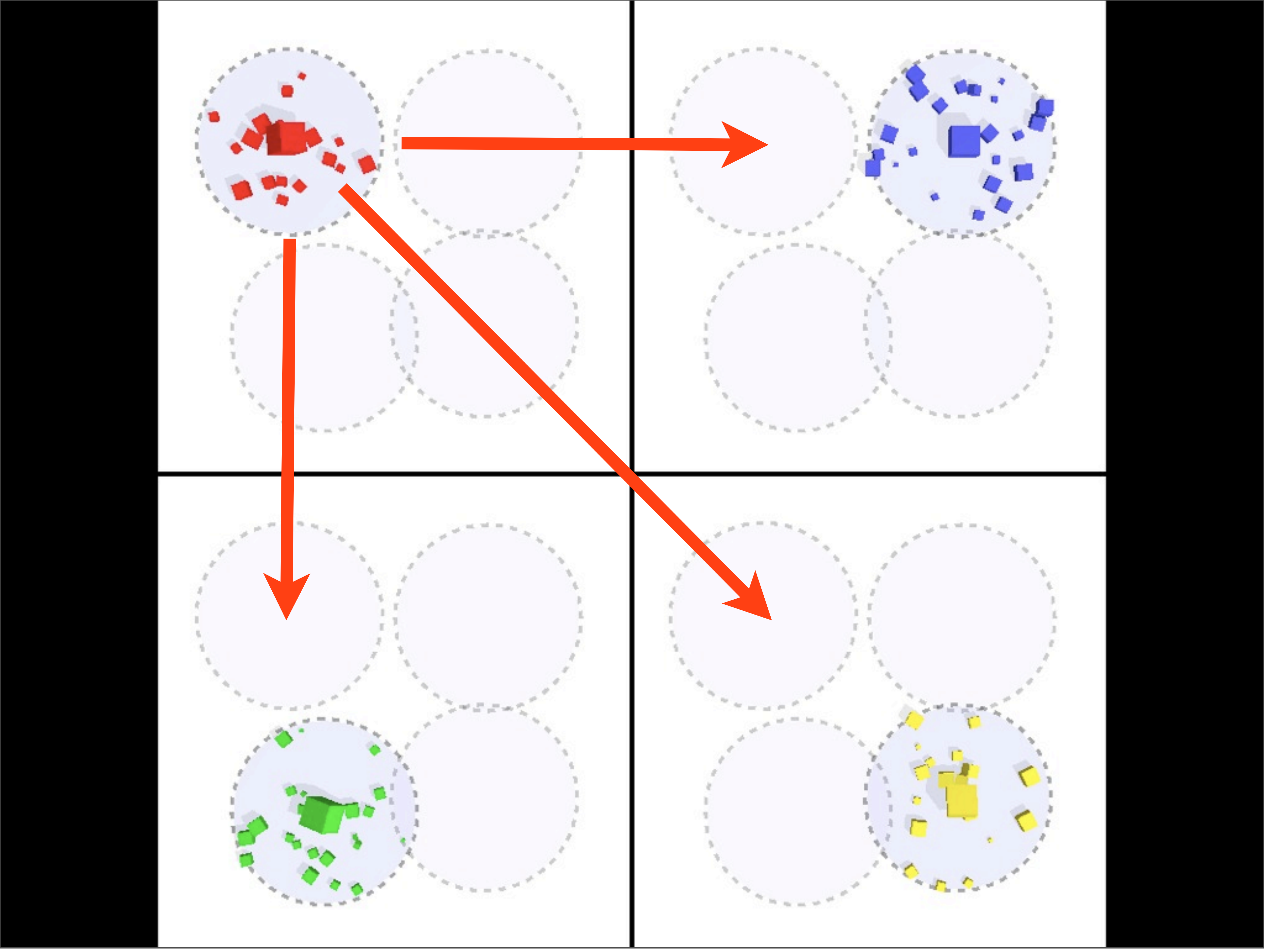
Now turn on sync by pressing F2 (it's off by default). This turns on everything up to and including interaction authority from previous section.

Note that the blue player still does not see the line cleared by the red player before he joined.

Roll the blue player across the red player's circle

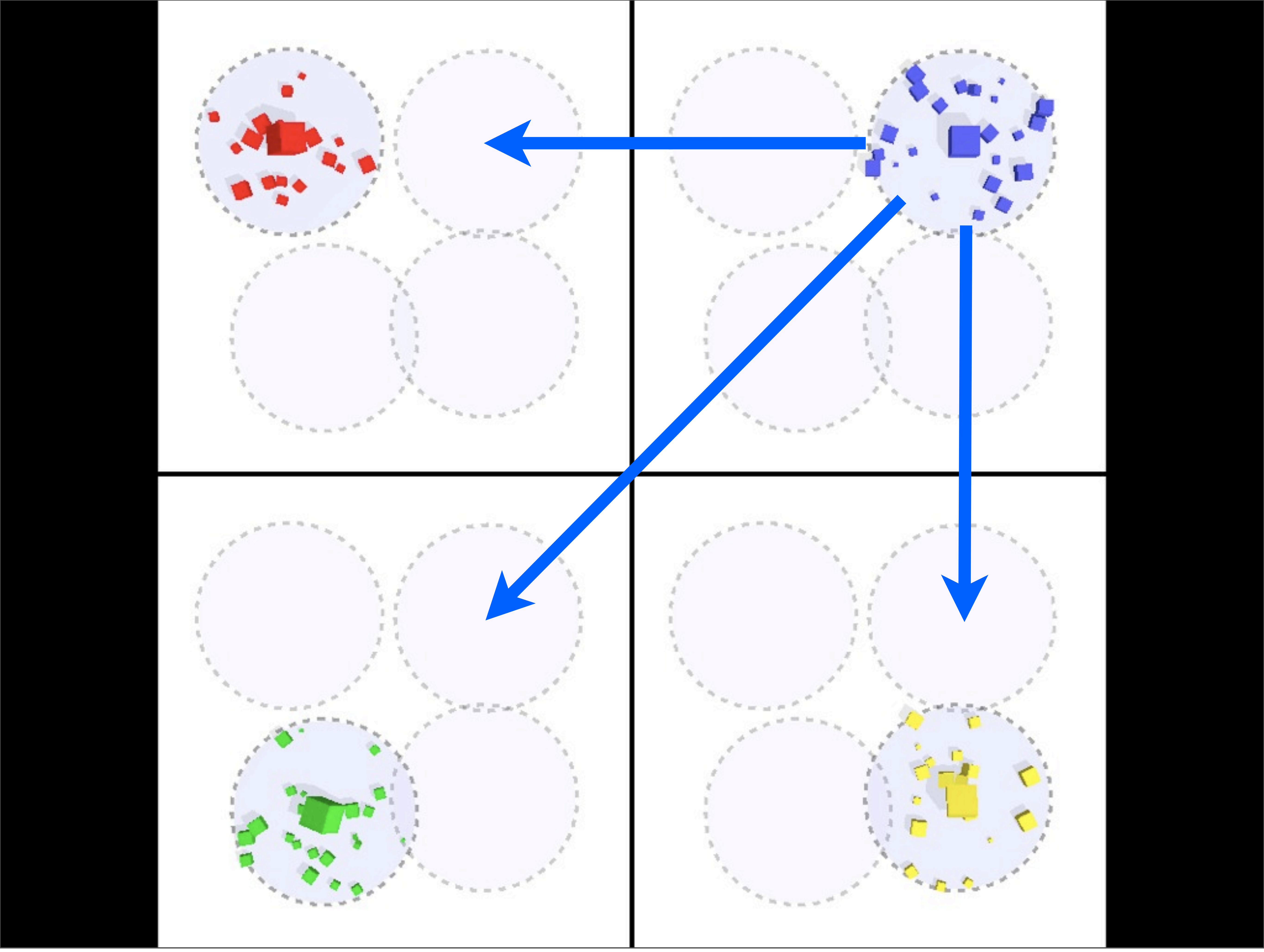
Look at the red player's state... the blue player has trashed the state of red player's world, undoing the line he cleared.

Why?



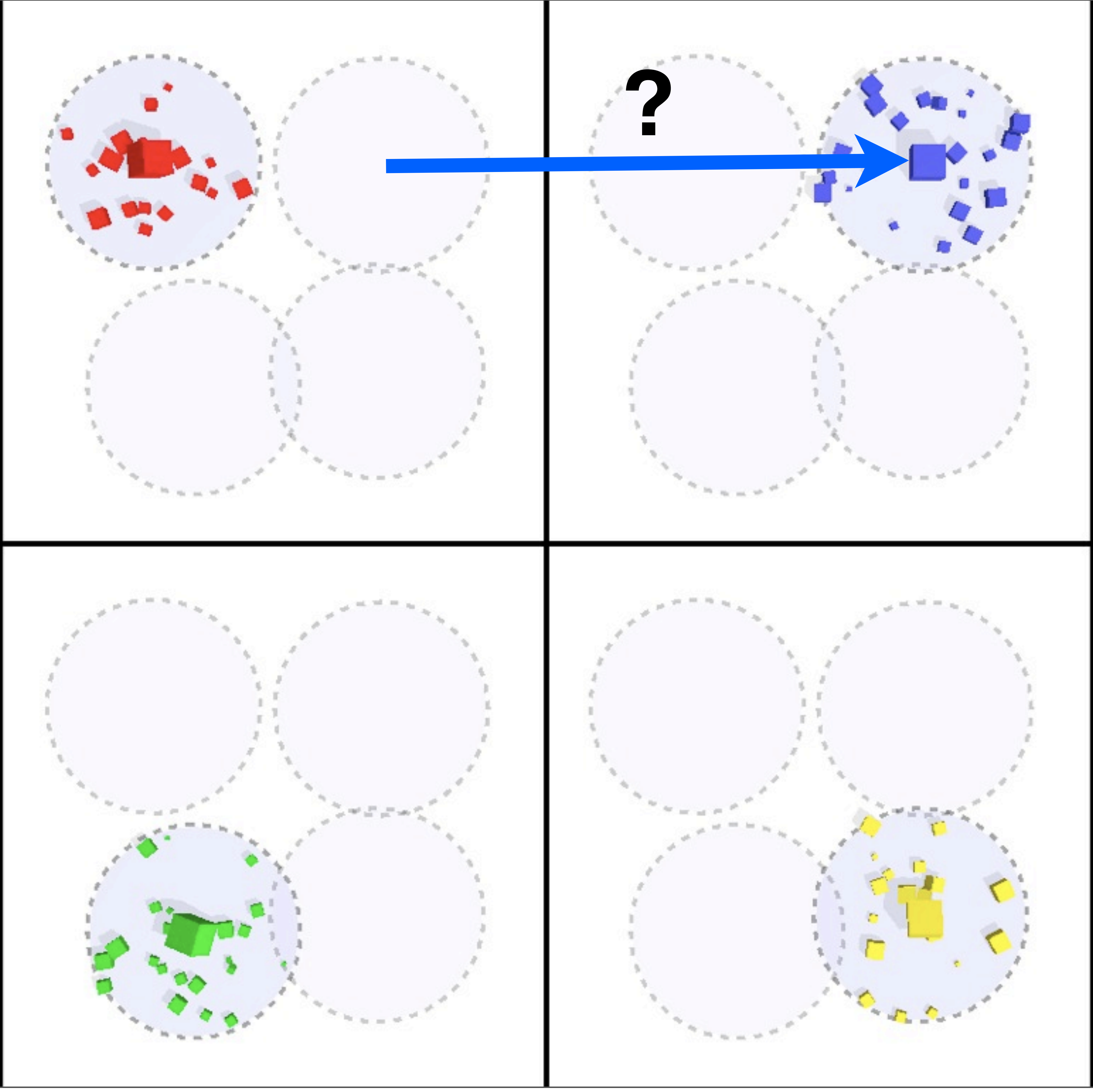
Tuesday, March 16, 2010

Active objects in red players machine are sent to blue



Tuesday, March 16, 2010

Blue player sends his active objects to red...



Tuesday, March 16, 2010

But no mechanism exists to send corrections from red player to blue in blue circle!

eg. the red player cannot correct the blue player when he is wrong

also the red player is always accepting updates from the blue player -- which causes the state to be trashed

how can we fix this?

Corrections

Fix incorrect state on late join

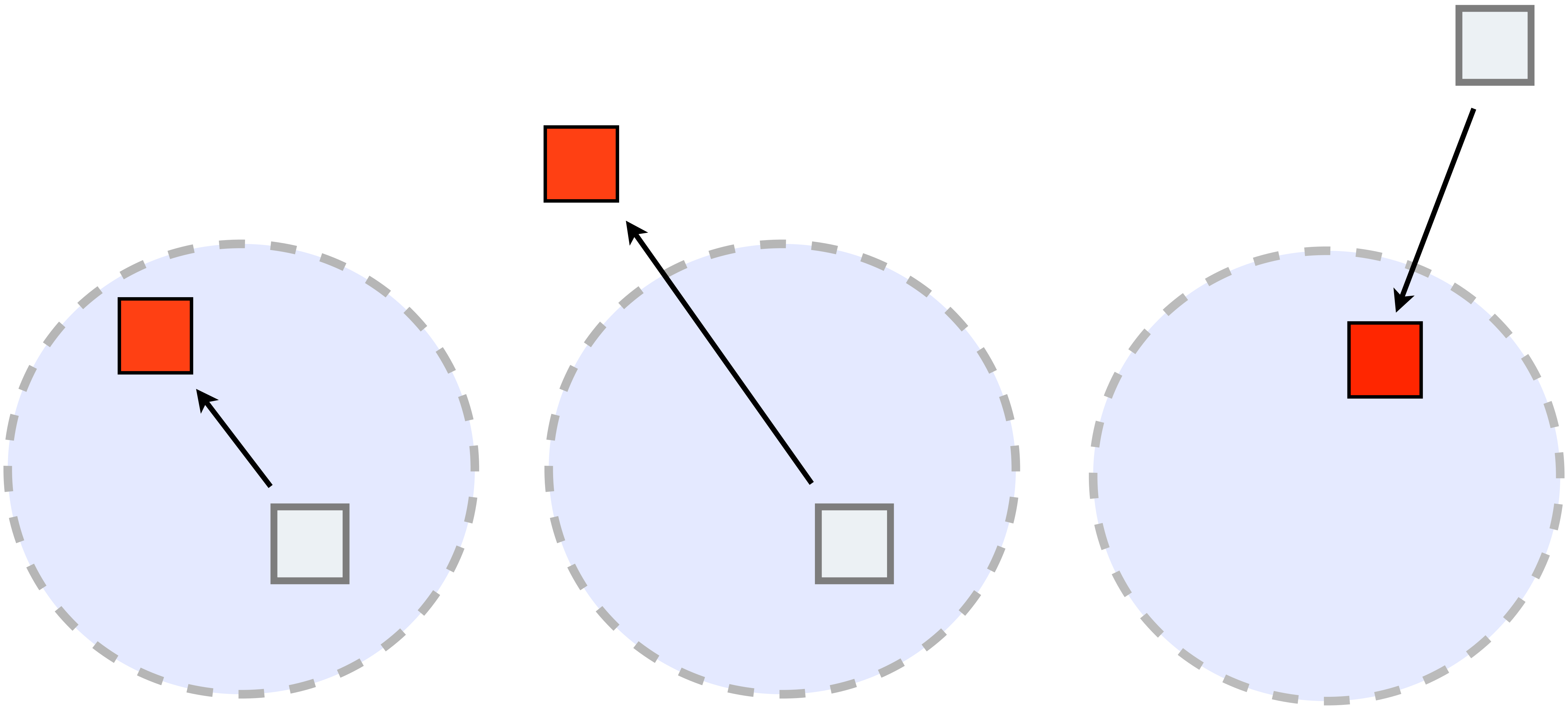
Tuesday, March 16, 2010

The solution is to add the concept of corrections.

A correction is the red player telling the blue player when he is incorrect about the state of objects in the world.

This way the red player informs the blue player, on a need-to-know basis -- about objects which had their state changed before the blue player joined the game.

Three Cases



Tuesday, March 16, 2010

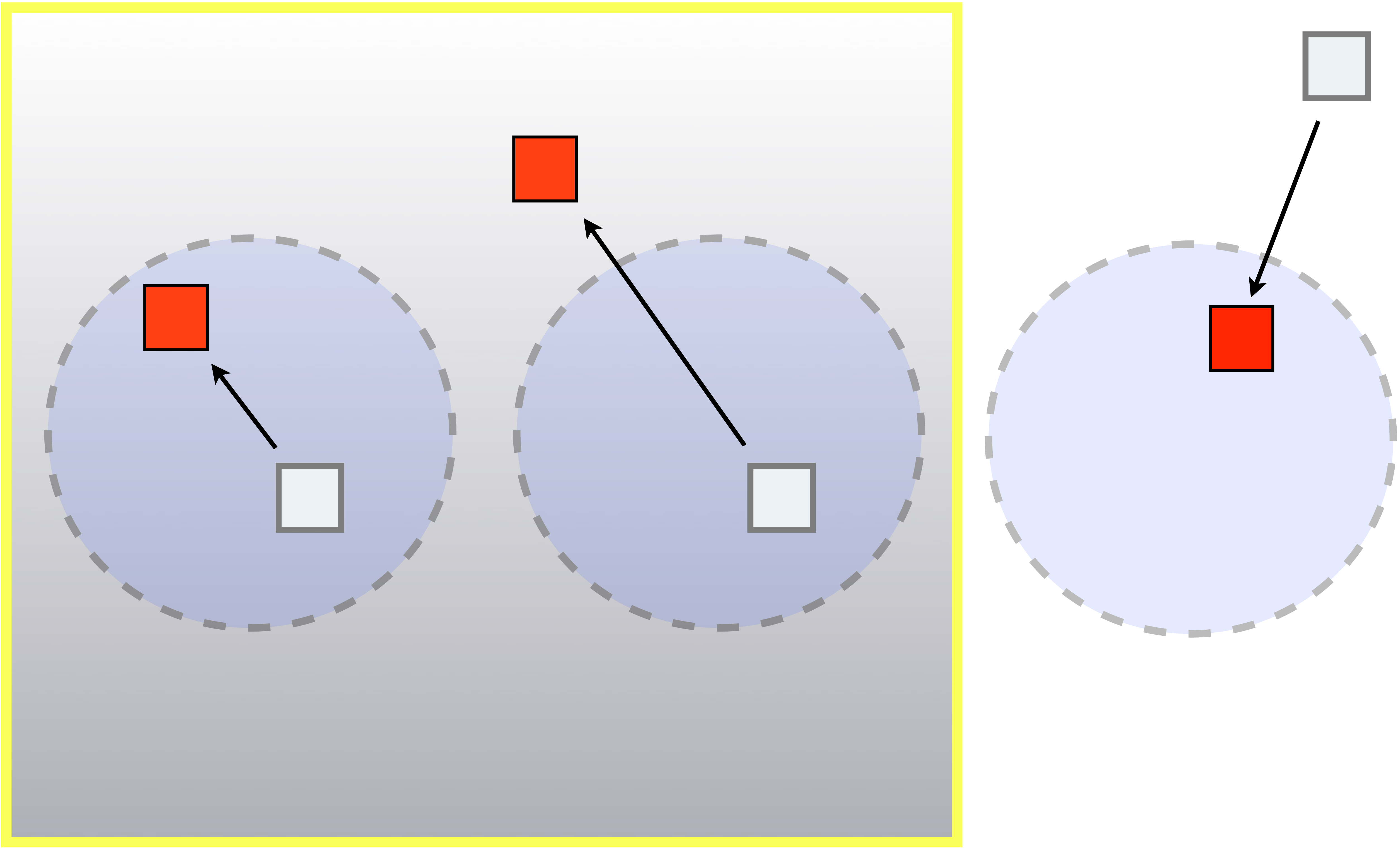
There are three cases, depending on what the red player has done and where the blue player circle is in the world

1. red player moves cube but it’s still inside blue player’s circle

2. red player moved cube outside blue player’s circle

3. red player moved a cube that was outside the circle, inside the blue player’s circle

First Two...

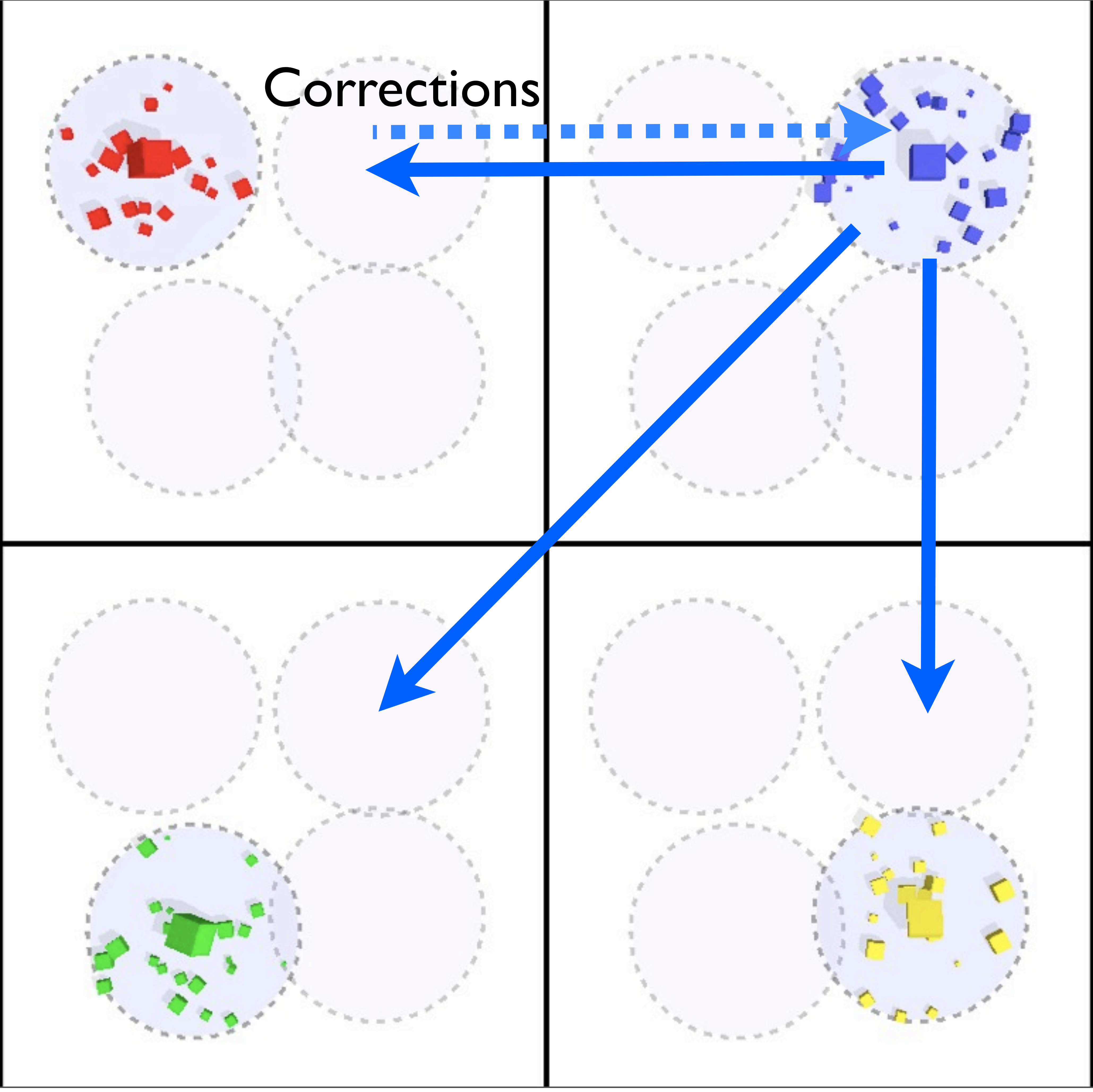


Tuesday, March 16, 2010

The first two cases are pretty easy

The third case is tricky...

Lets look at two easy cases first because they have the same solution



Tuesday, March 16, 2010

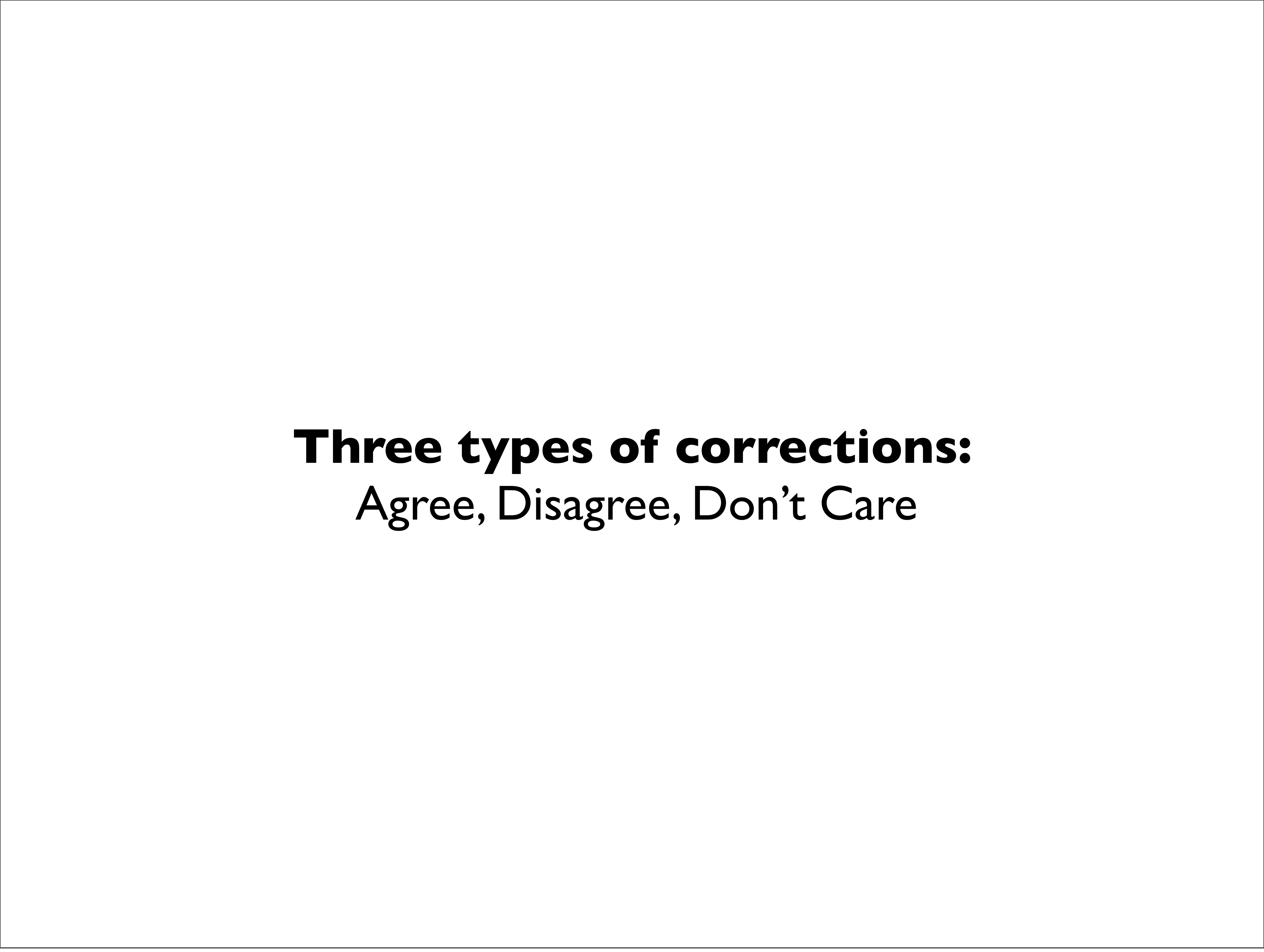
Note that so far the red player is continually sending his active objects to the blue player

And the blue player is sending his active objects to the red player (this is what trashed state)

But there is no mechanism for telling the blue player when the state he is sending to the red player is incorrect

Here is what we do:

Each time the red player receives state for a cube from the blue player, he looks at it and queues up a correction if that state is incorrect



Three types of corrections:

Agree, Disagree, Don’t Care

Tuesday, March 16, 2010

In fact it’s more complicated than this, the blue player may be sending the red player state for objects which the red player has never activated, or perhaps some objects that the red player agrees are in the correct place

so we don’t always say “nonono, it should be over here”

For each blue object state coming in from the player the red player inspects it and replies: “agree, disagree or don’t care”

agree means, ok i confirm that the object is actually where you think it is (within some tolerance)

disagree means - nonono, you have it all wrong it’s over HERE instead (position and orientation are included to snap to)

don’t care means -- i have never activated this object before and therefore I have no opinion where it should be!

Confirmed bit

Per-Object, Per-Player

Avoids trashing state

Tuesday, March 16, 2010

It is even more complicated however, because not only does the red player need to correct the blue player when state is wrong -- the red player also needs to know when it is safe to accept state from the blue player

For players above red player objects are initially unconfirmed (red is confirmed by default, he started the game nobody can join before him!)

In the packet blue sends to red, for each object he includes “confirmed” true/false

If the state is unconfirmed the red player queues up a correction for the blue player to be included in a future packet

Once the blue player receives “agree”, “disagree” or “dontcare” for the object -- he sets the confirmed bit to true

The blue player only needs confirmation from the red player before he sets bit to true for object.

Green player needs confirmation from red and blue.

Yellow needs confirmation from red, blue and green.

This seems simple but it is actually quite complicated in practice especially when generalized to four players

Please look at the source code for implementation details.

DEMO

demo showing working corrections, red clears swath -- blue sees corrections applied as he moves along the swath, does not trash red player’s state

-----

To reproduce this in the demo, press ALT-4 to select corrections demo (or ALT-5 in MIGS’09 win32 build)

Then press tab once to see other player circles

Hold space then move the red player with arrow keys to clear a horizontal line, including right through the blue player’s circle

Return the red player back to his initial position (press 6 for overhead view, line him up in top-left)

Press enter twice to go to quadscreen.

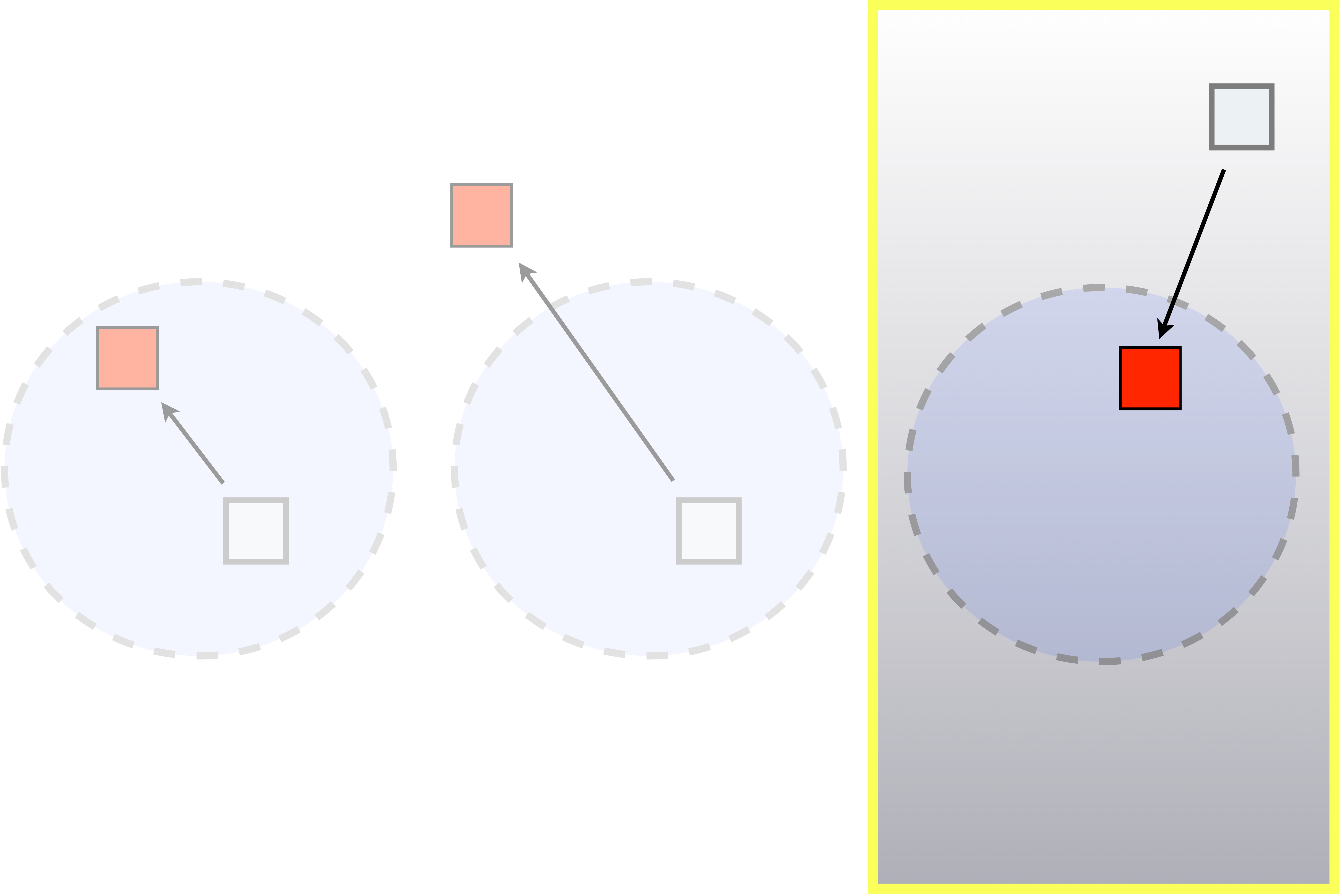
There is no sync on right now, note that the blue player does not see the line cleared by red

Now turn on sync + corrections by pressing **F3** (it’s off by default). This turns on everything up to and including interaction authority from previous section and the corrections we just talked about.

Note that as the blue player moves around the world the corrections stream in and the red player’s swath is cleared in the world.

Also note that the blue player does not trash the red player’s world state.

Three Cases



Tuesday, March 16, 2010

The third case is very tricky!

The first two cases can be handled simply by looking for incorrect object state coming in the stream of objects sent from blue -> red

But the third case cannot... This is the case where the red player moved an object into the blue player’s circle before blue joined

Of course, blue does not know the object is inside his circle, so he is not sending state for it to the red player... normal corrections do not work

DEMO

demo shows red player rolling up a large katamari of cubes then dumping them in the blue players circle -- blue player joins and the blue simulation does not see all cubes dumped in his circle, only a few...

Why?

To reproduce this in the demo, press ALT-4 to select authority demo (or ALT-5 in MIGS'09 win32 build)

Then press tab once to see other player circles

Now as red player hold z and roll up a big katamari of cubes then roll over next to the blue player and dump the cubes in there (let go of z)

Roll the red player back to his starting point top-left

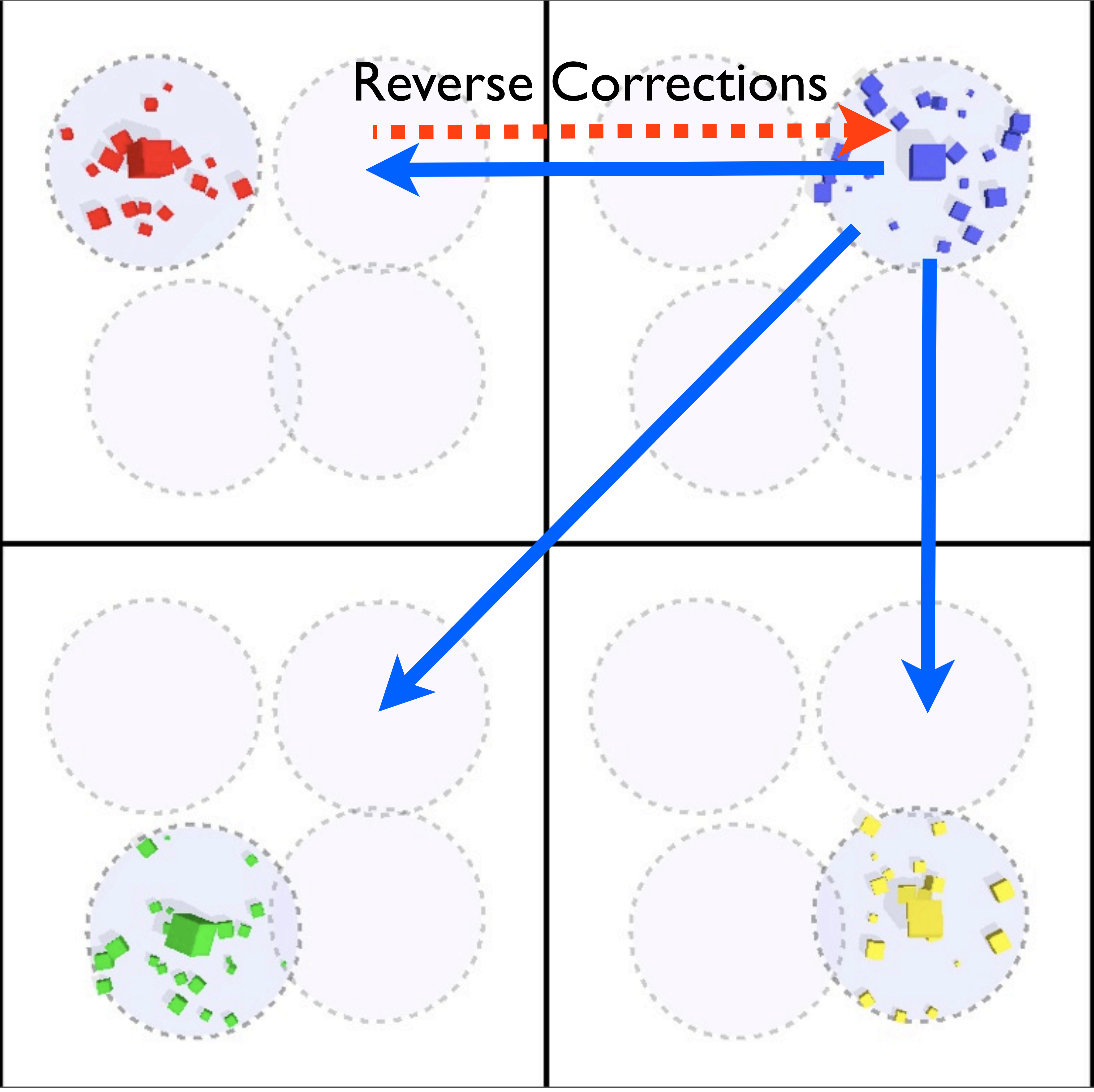
Press enter twice to go to quadscreen.

There is no sync on right now, note that the blue player does not see any changes made by red

Now turn on sync + corrections by pressing **F3** (it's off by default). This turns on everything up to and including interaction authority from previous section <u>and</u> the corrections we just talked about.

The blue player does not see the big pile of cubes the red player dumped in his circle

Why?



Tuesday, March 16, 2010

What we need is a new concept of “reverse corrections”

Red player maintains set of objects he “expects” the blue player should be sending to him, eg. the objects he thinks are in the blue player’s circle. He does not simulate these objects, he just tracks that the should be active for the blue player

Then red watches the stream of incoming objects from the blue player

If he does not receive any update for an object which he thinks is inside the circle, red queues up a “reverse correction” to be sent back to blue player

This functions just the same way as a normal correction except it is the inverse -- we are sending corrections for objects which we do not receive instead of sending a correction for object state we did receive

Summary

Hide latency with authority scheme

Handle late join using corrections

Suitable for cooperative games

This technique is good for networking a cooperative game where players interact with physically simulated objects

And where these objects typically return to being at rest after being interacted with, and where in general -- the world is large enough for players to be interacting with different islands of objects

In this common case latency is completely hidden via authority scheme

Late join can be supported (in a very complicated fashion) using corrections

It is suitable only for cooperative games, because it is very prone to cheating (each player tells other players the state of the world...)

But for a large subset of games on closed platforms like PSN you can get a very cool game where players work together to solve physics puzzles in an open world --- who will make this game first?

Will it be you?

Questions?

Tuesday, March 16, 2010

There are many other ways to skin this cat

There is a variant for authority scheme which uses a dedicated server, see the insomniac “Sync Host” paper on their R&D website

In this client/server technique the authority scheme may still be used but the problem of late join is much easier to solve

You may also look into deterministic lockstep synchronization if you have low player counts and too much state to send. You just need to make sure your physics engine is deterministic.

See this page for more info http://gafferongames.com/networking-for-game-programmers/floating-point-determinism/

If you use this technique in a shipping game or have any questions about it -- please contact me I’d love to hear from you

Thank you

Thanks for reading!

Glenn Fiedler

www.gafferongames.com

twitter

@gafferongames

Please visit www.gafferongames.com for more information about game physics and networking

Here you can also find a link to download an executable and source code for the demo used in this talk

And make sure you follow me on twitter: @gafferongames

<u>Have fun with the source code!</u>