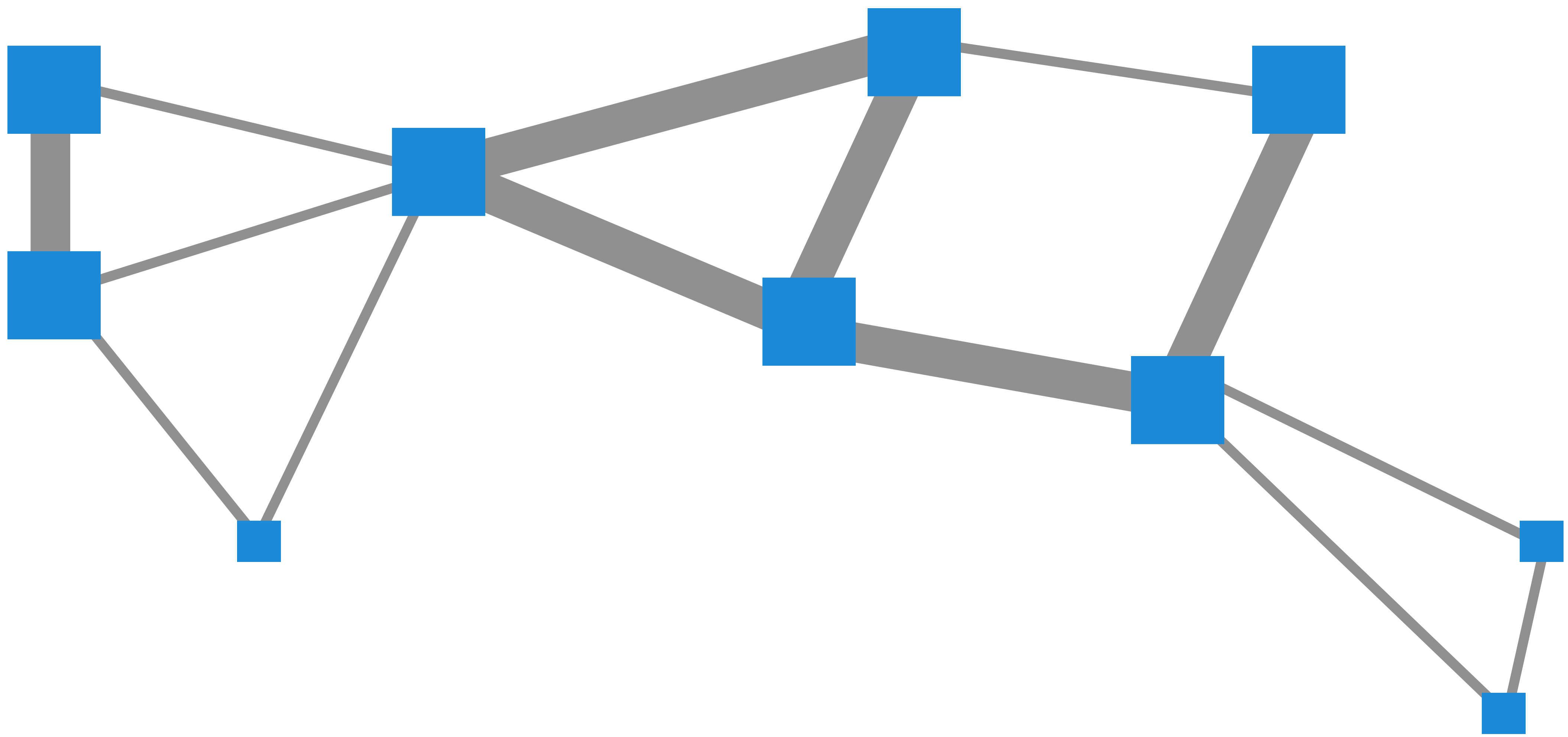
Living with Big Data: Challenges and Opportunities

Jeff Dean, Sanjay Ghemawat
Google

Joint work with many collaborators

Computational Environment

- Many datacenters around the world

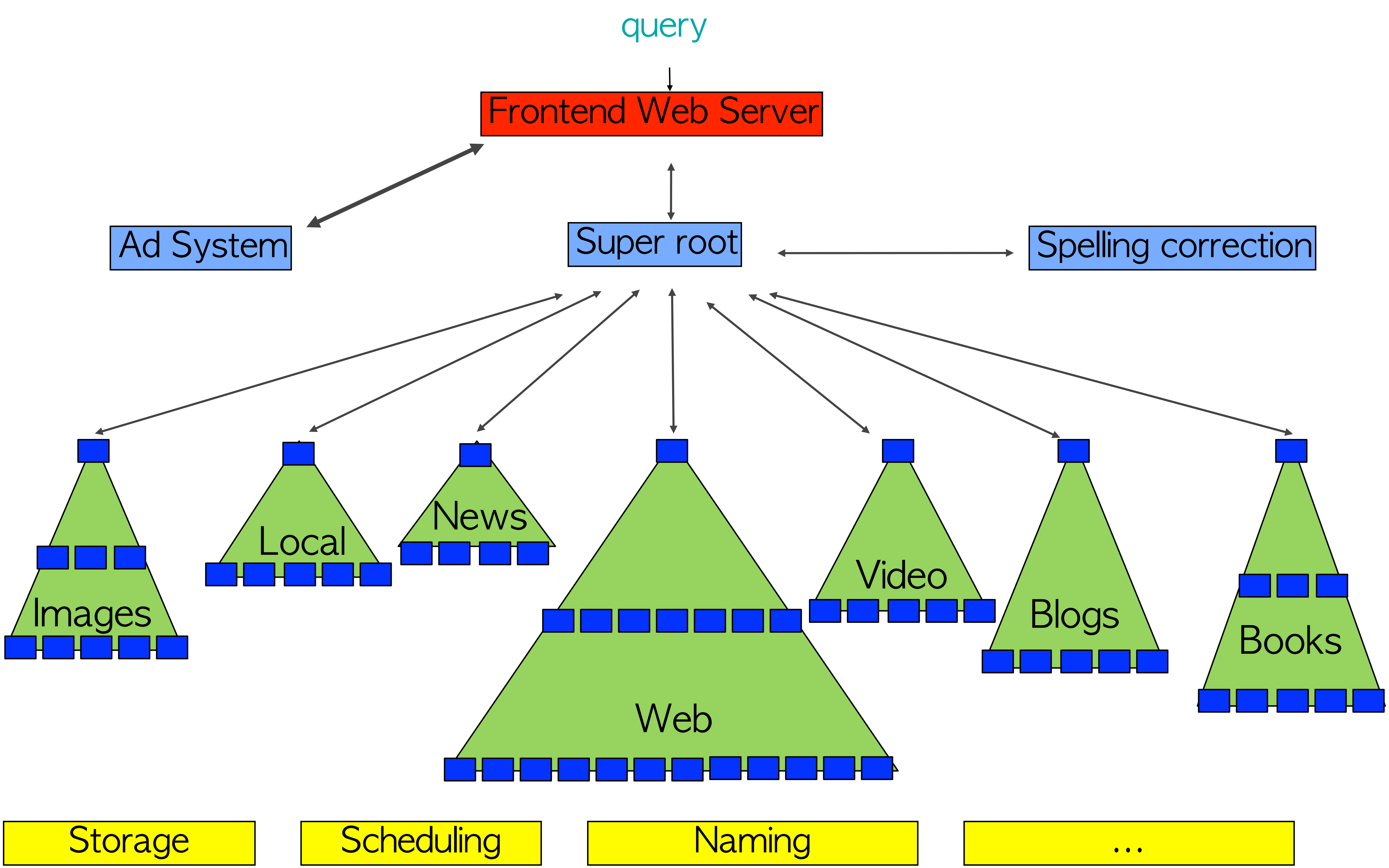


Zooming In...

Google

Zooming In...

Decomposition into Services



Communication Protocols

- Example:
  - Request: `query: “ethiopiaan restaurnts”`
  - Response: list of (corrected query, score) results

    ```
    correction { query: “ethiopian restaurants” score: 0.97 }
    correction { query: “ethiopia restaurants” score: 0.02 }
    ...
    ```

- Benefits of structure:
  - easy to examine and evolve (add `user_language` to request)
  - language independent
  - teams can operate independently

- We use Protocol Buffers for RPCs, storage, etc.
  - http://code.google.com/p/protobuf/

Google

The Horrible Truth...

Typical first year for a new cluster:

~1 network rewiring (rolling ~5% of machines down over 2-day span)

~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)

~5 racks go wonky (40-80 machines see 50% packetloss)

~8 network maintenances (4 might cause ~30-minute random connectivity losses)

~12 router reloads (takes out DNS and external vips for a couple minutes)

~3 router failures (have to immediately pull traffic for an hour)

~dozens of minor 30-second blips for dns

~1000 individual machine failures

~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

The Horrible Truth...

Typical first year for a new cluster:

~1 network rewiring (rolling ~5% of machines down over 2-day span)

~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)

~5 racks go wonky (40-80 machines see 50% packetloss)

~8 network maintenances (4 might cause ~30-minute random connectivity losses)

~12 router reloads (takes out DNS and external vips for a couple minutes)

~3 router failures (have to immediately pull traffic for an hour)

~dozens of minor 30-second blips for dns

~1000 individual machine failures

~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

- Reliability/availability must come from software!

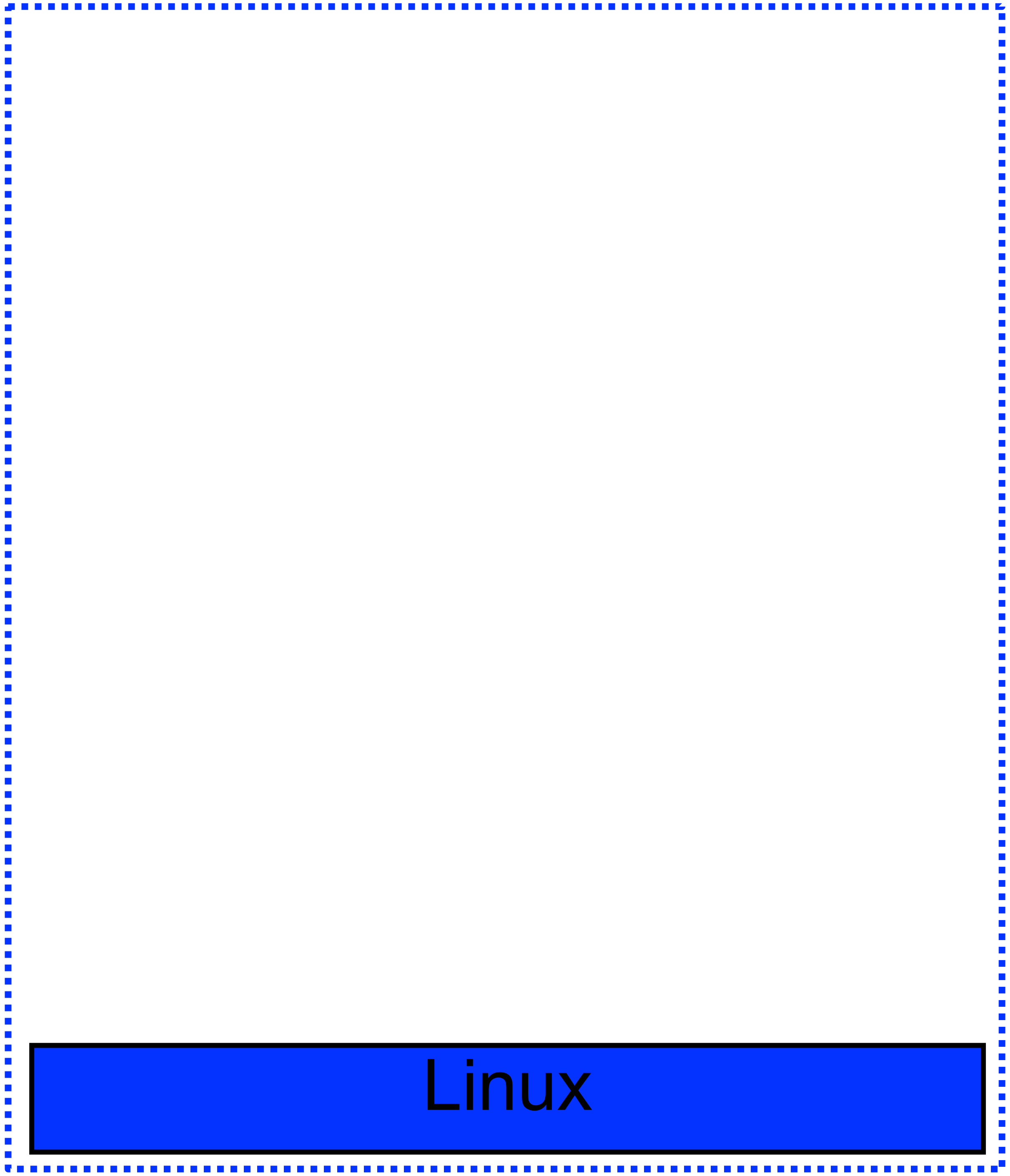
Google

Replication

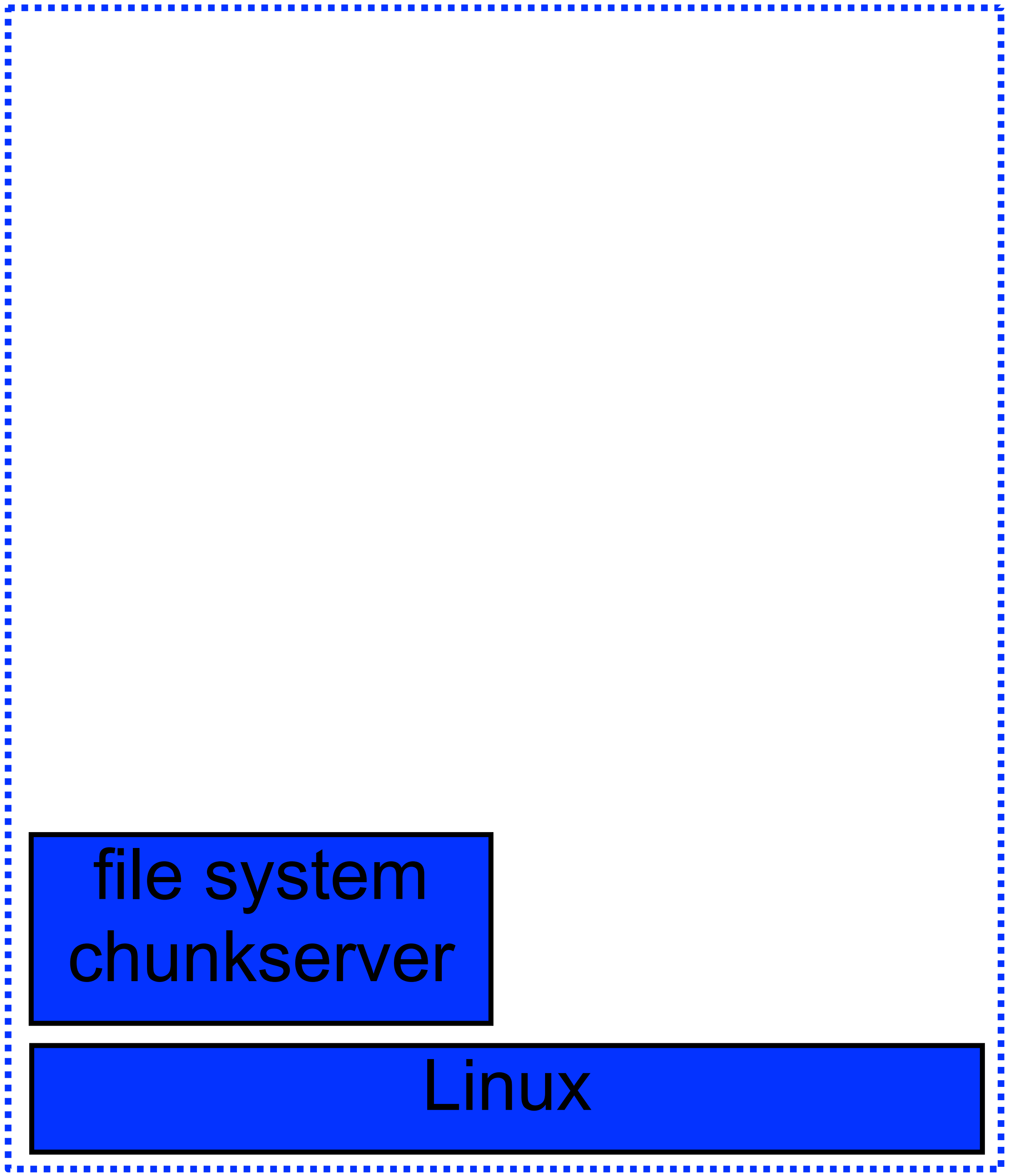
- Data loss
  - replicate the data on multiple disks/machines (GFS/Colossus)
- Slow machines
  - replicate the computation (MapReduce)
- Too much load
  - replicate for better throughput (nearly all of our services)
- Bad latency
  - utilize replicas to improve latency
  - improved worldwide placement of data and services

Google

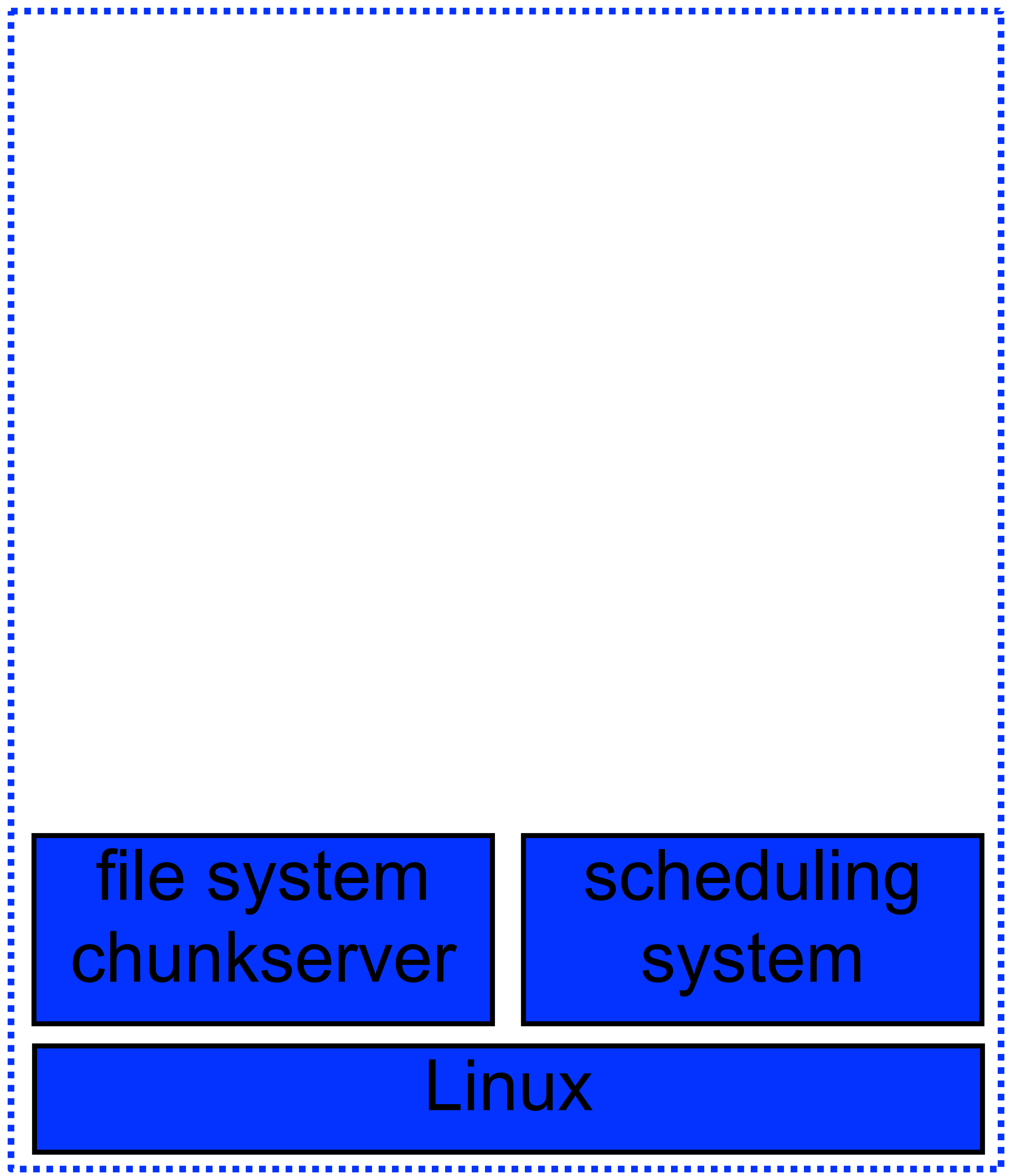
Shared Environment



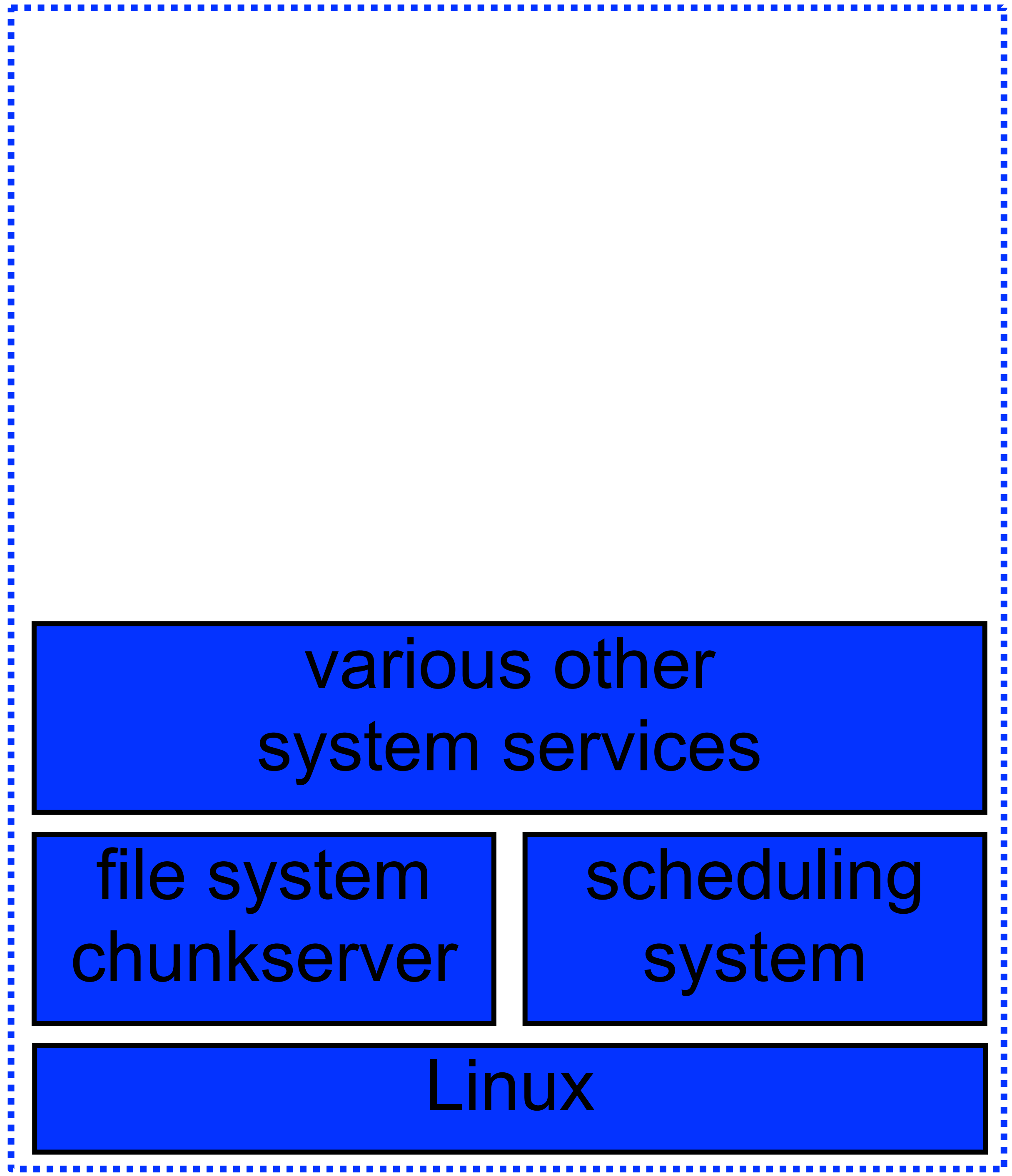
Shared Environment



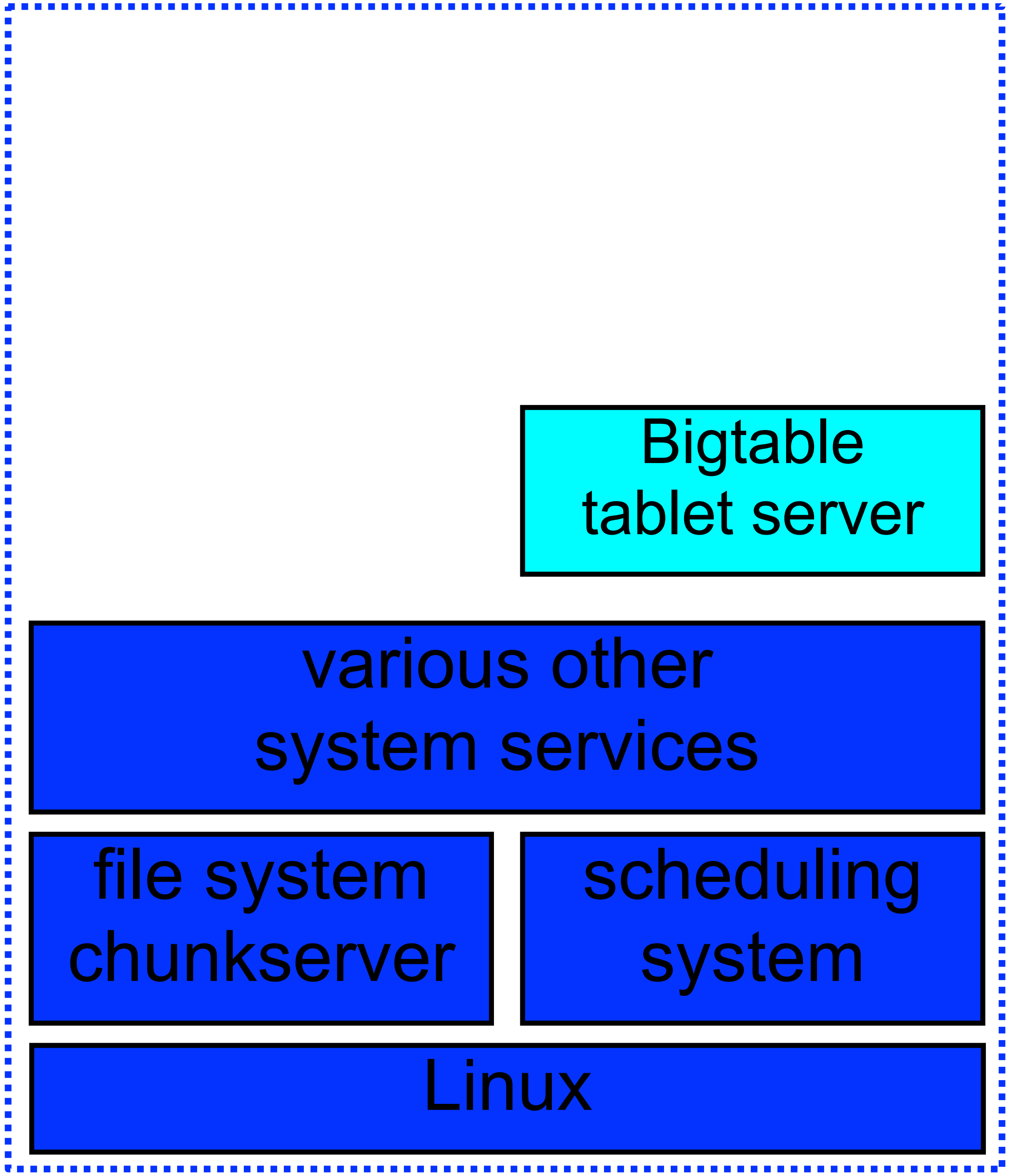
Shared Environment



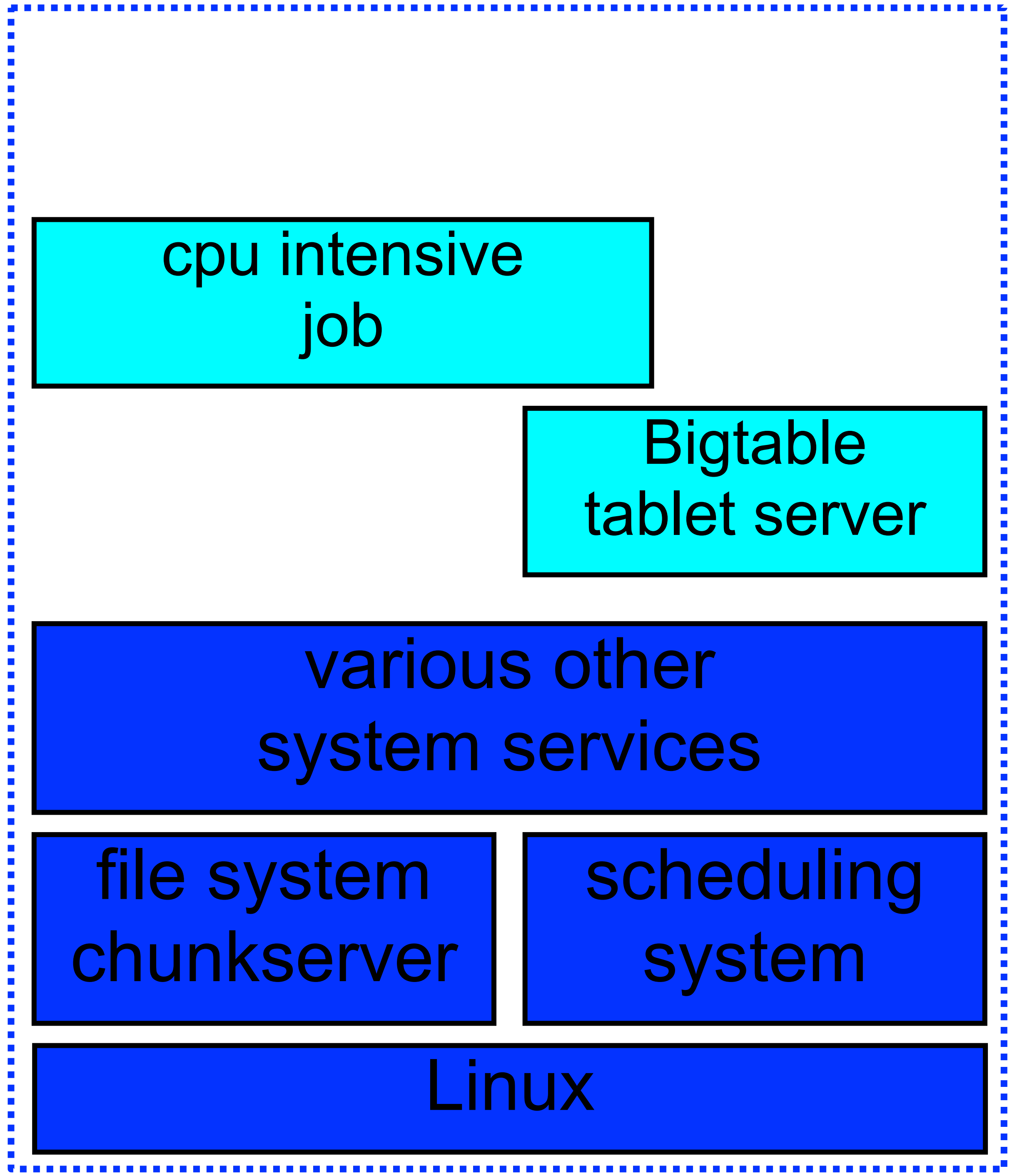
Shared Environment



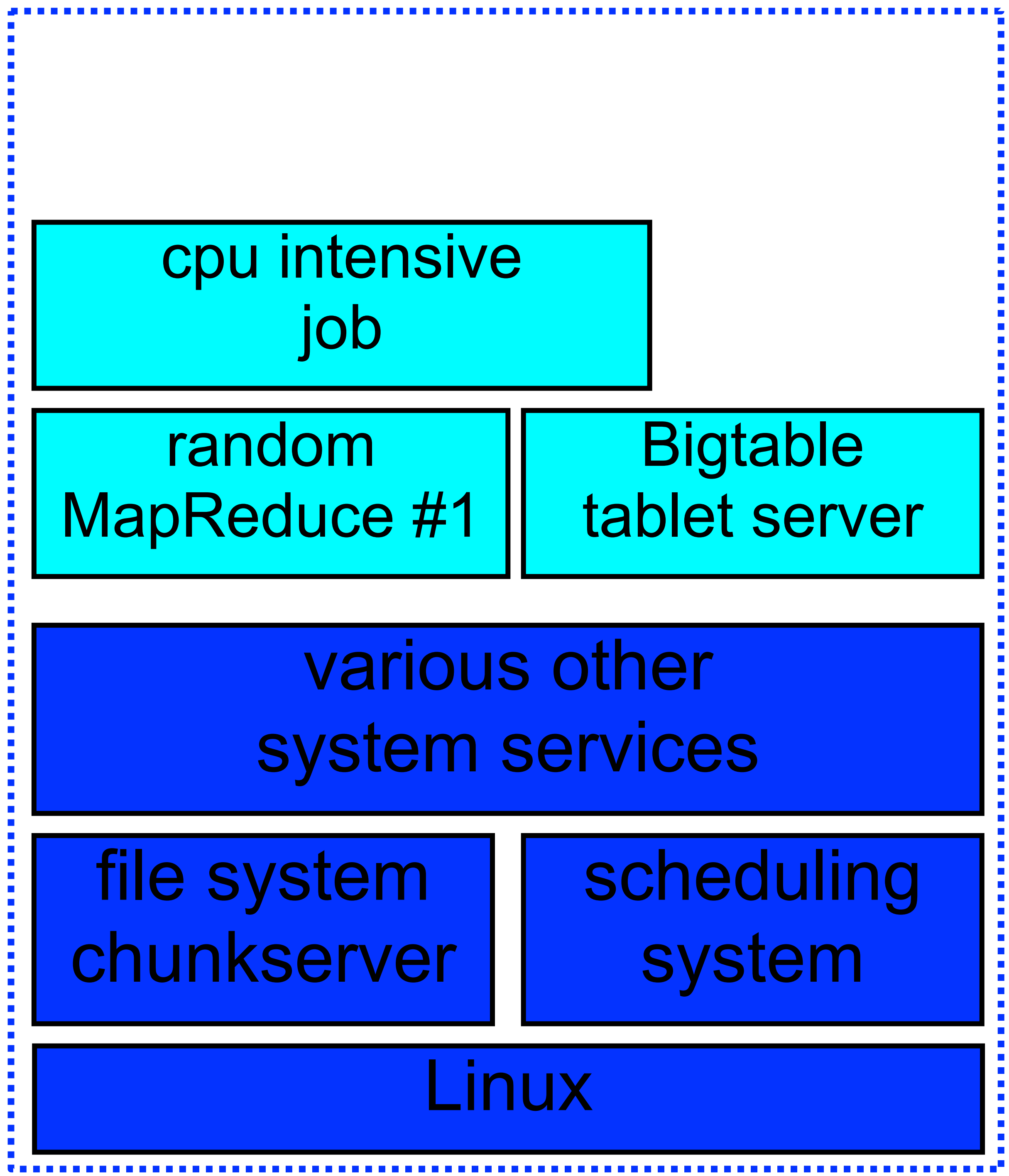
Shared Environment



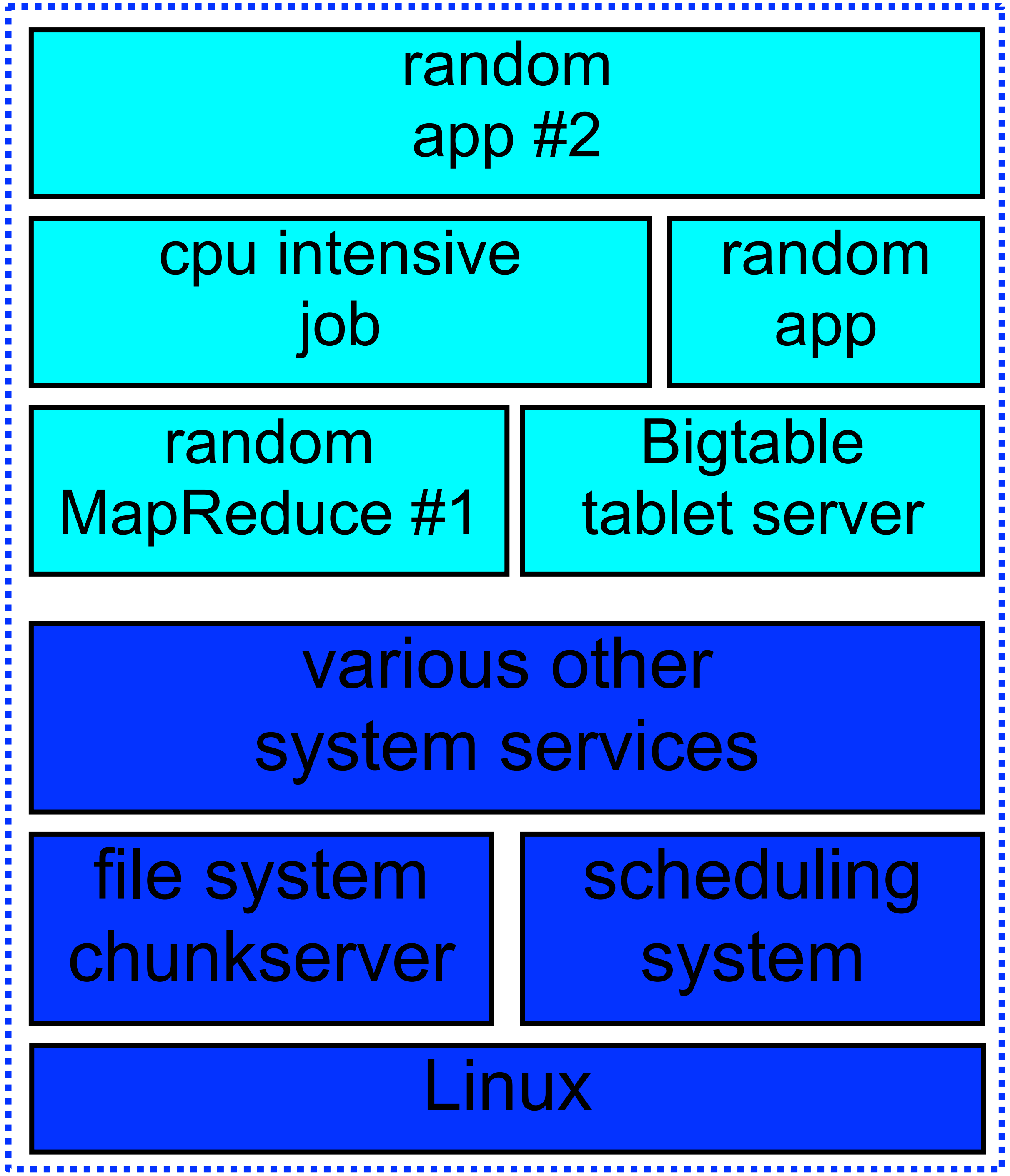
Shared Environment



Shared Environment



Shared Environment



Google

Shared Environment

- Huge benefit: greatly increased utilization

- ... but hard to predict effects increase variability
  - network congestion
  - background activities
  - bursts of foreground activity
  - not just your jobs, but everyone else's jobs, too
  - not static: change happening constantly

- Exacerbated by large fanout systems

Google

The Problem with Shared Environments

The Problem with Shared Environments

Google

The Problem with Shared Environments

- Server with 10 ms avg. but 1 sec 99%ile latency
  - touch 1 of these: 1% of requests take ≥1 sec
  - touch 100 of these: 63% of requests take ≥1 sec

Google

Tolerating Faults vs. Tolerating Variability

- Tolerating faults:
  - rely on extra resources
    - RAIDed disks, ECC memory, dist. system components, etc.
  - ***make a reliable whole out of unreliable parts***

- Tolerating variability:
  - use these same extra resources
  - ***make a predictable whole out of unpredictable parts***

- Times scales are very different:
  - variability: 1000s of disruptions/sec, scale of **milliseconds**
  - faults: 10s of failures per day, scale of **tens of seconds**

Google

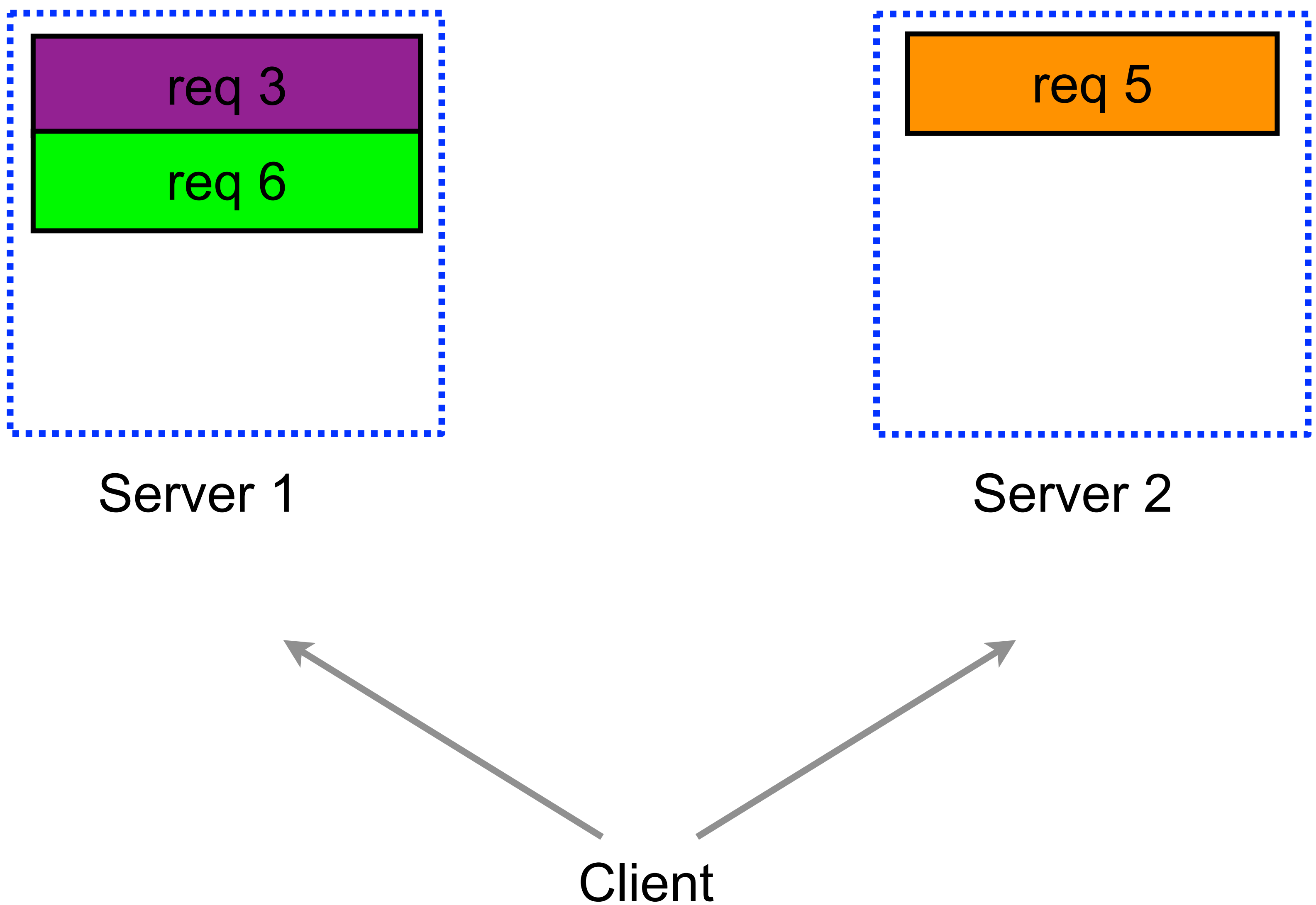
Latency Tolerating Techniques

- Cross request adaptation
  - examine recent behavior
  - take action to improve latency of future requests
  - typically relate to balancing load across set of servers
  - time scale: 10s of seconds to minutes
- Within request adaptation
  - cope with slow subsystems in context of higher level request
  - time scale: right now, while user is waiting
- Many such techniques

  [*The Tail at Scale*, Dean & Barroso, to appear in CACM late 2012/early 2013]

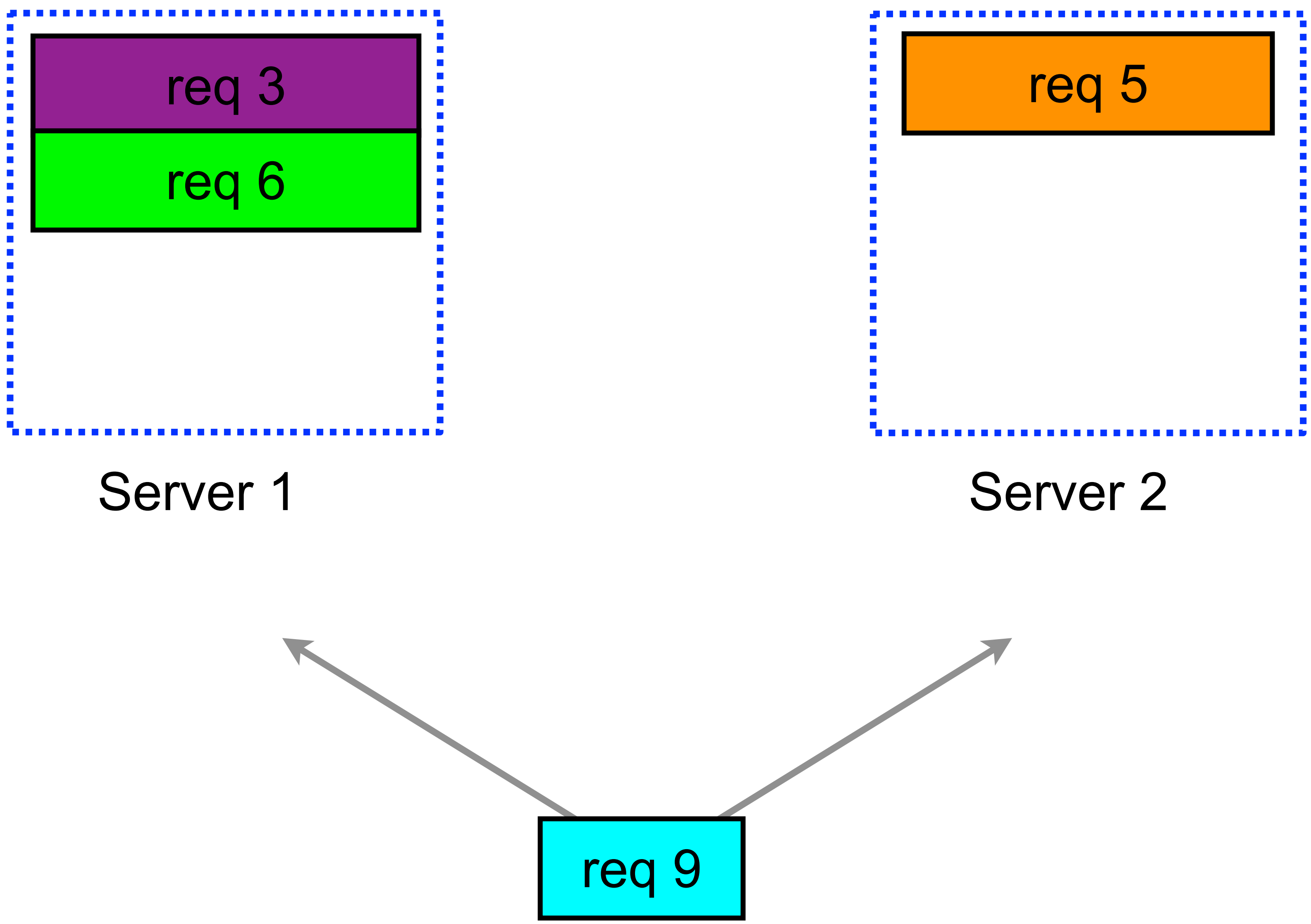
Google

Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

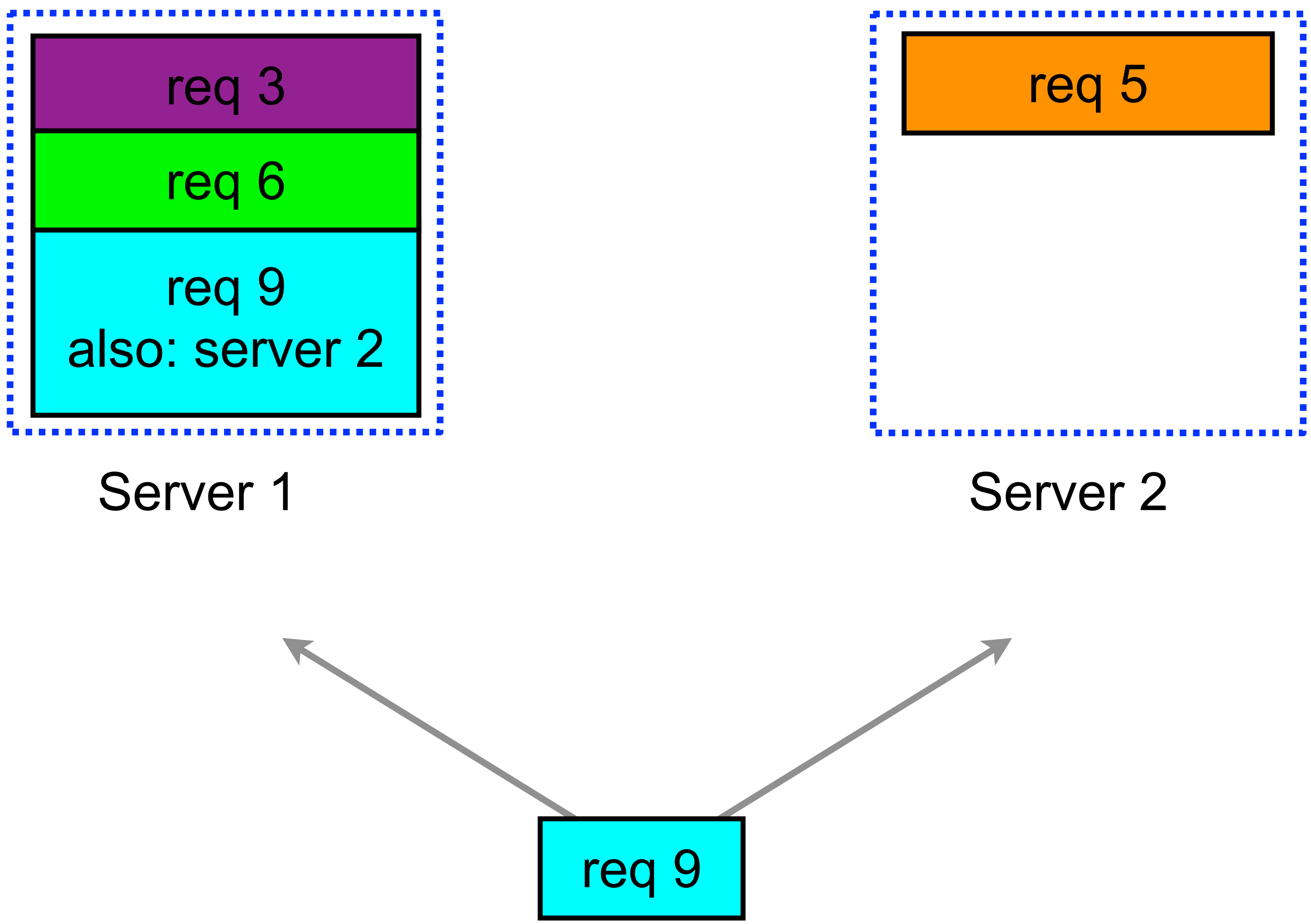
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Google

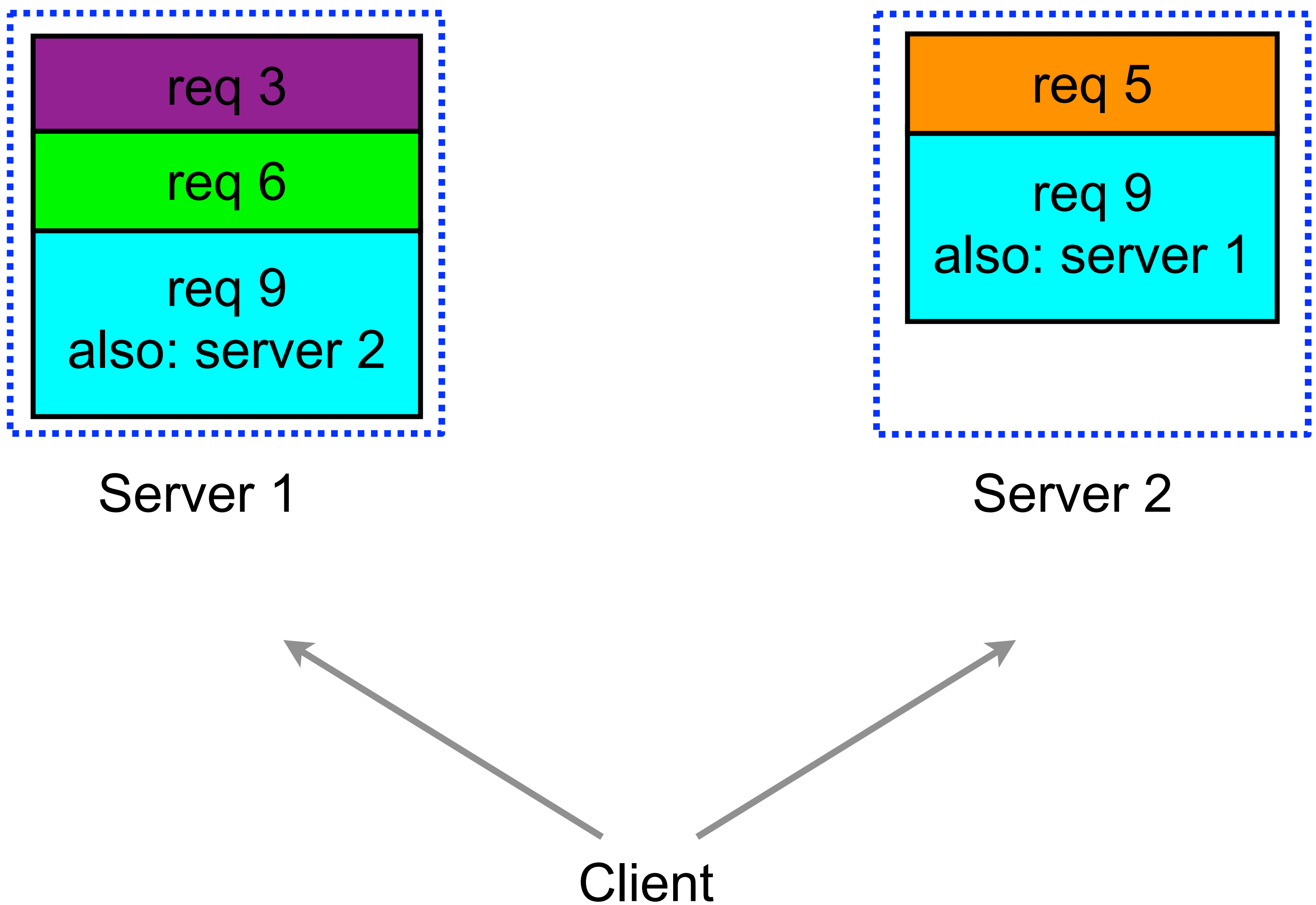
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent Google

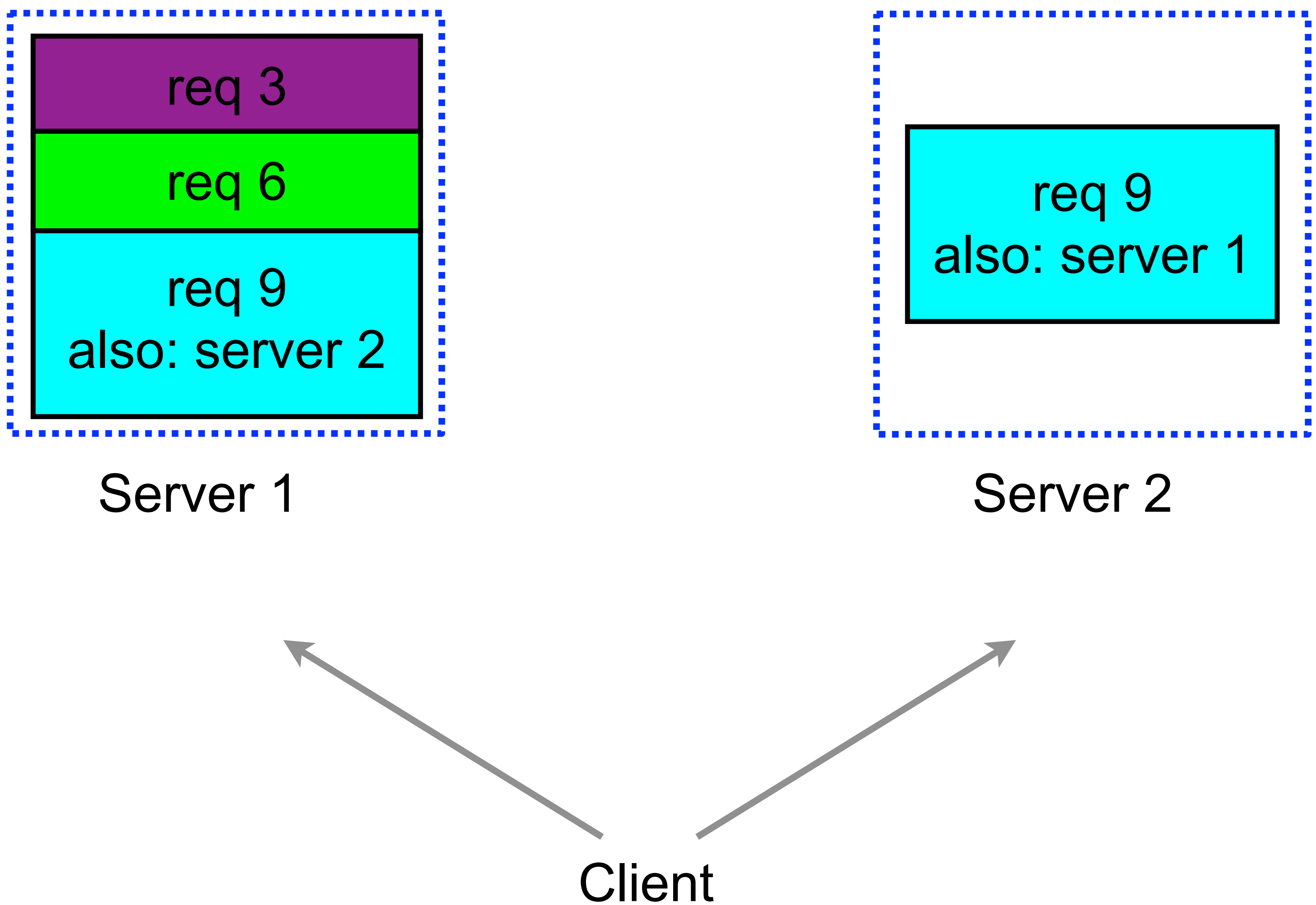
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent Google

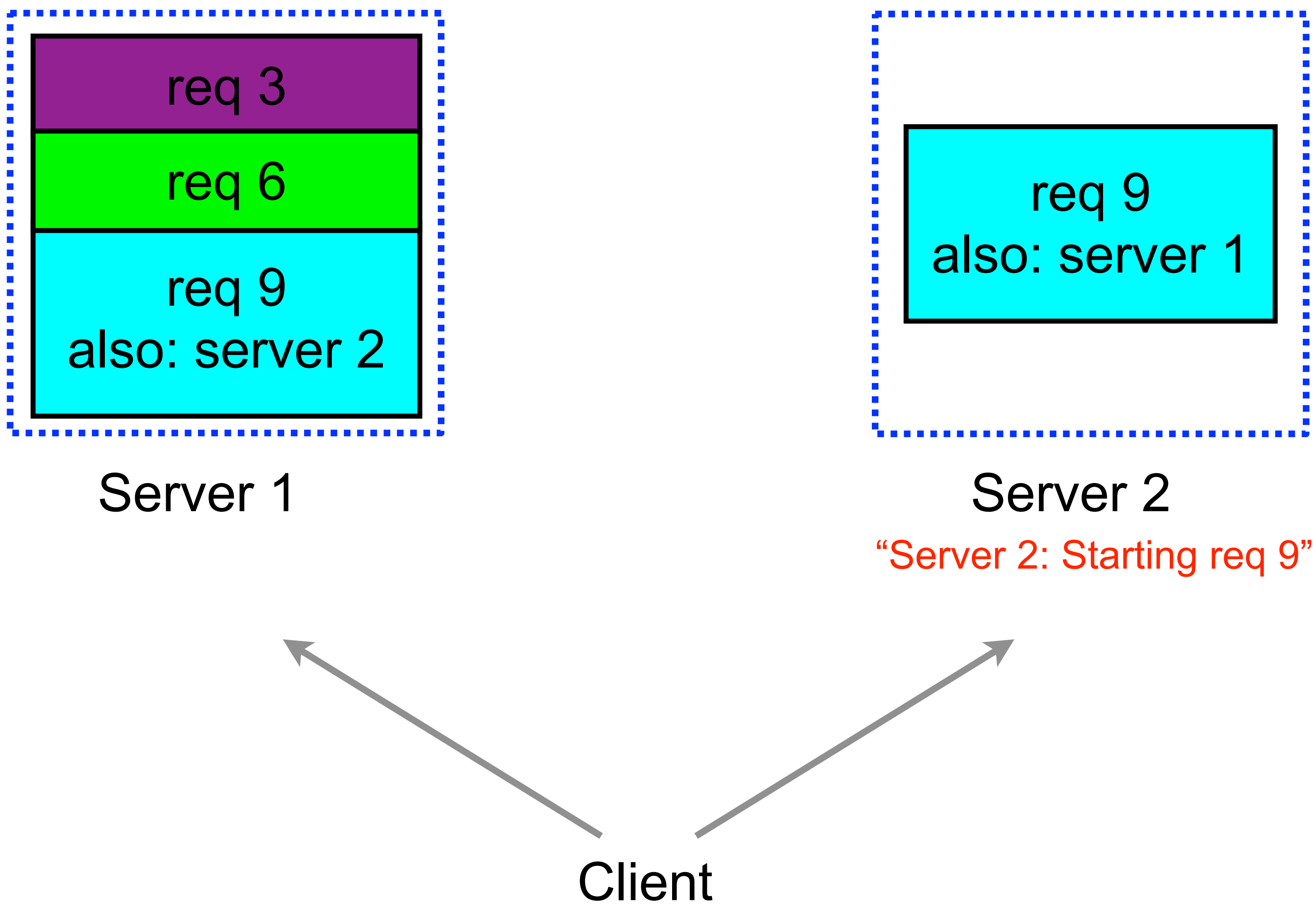
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent

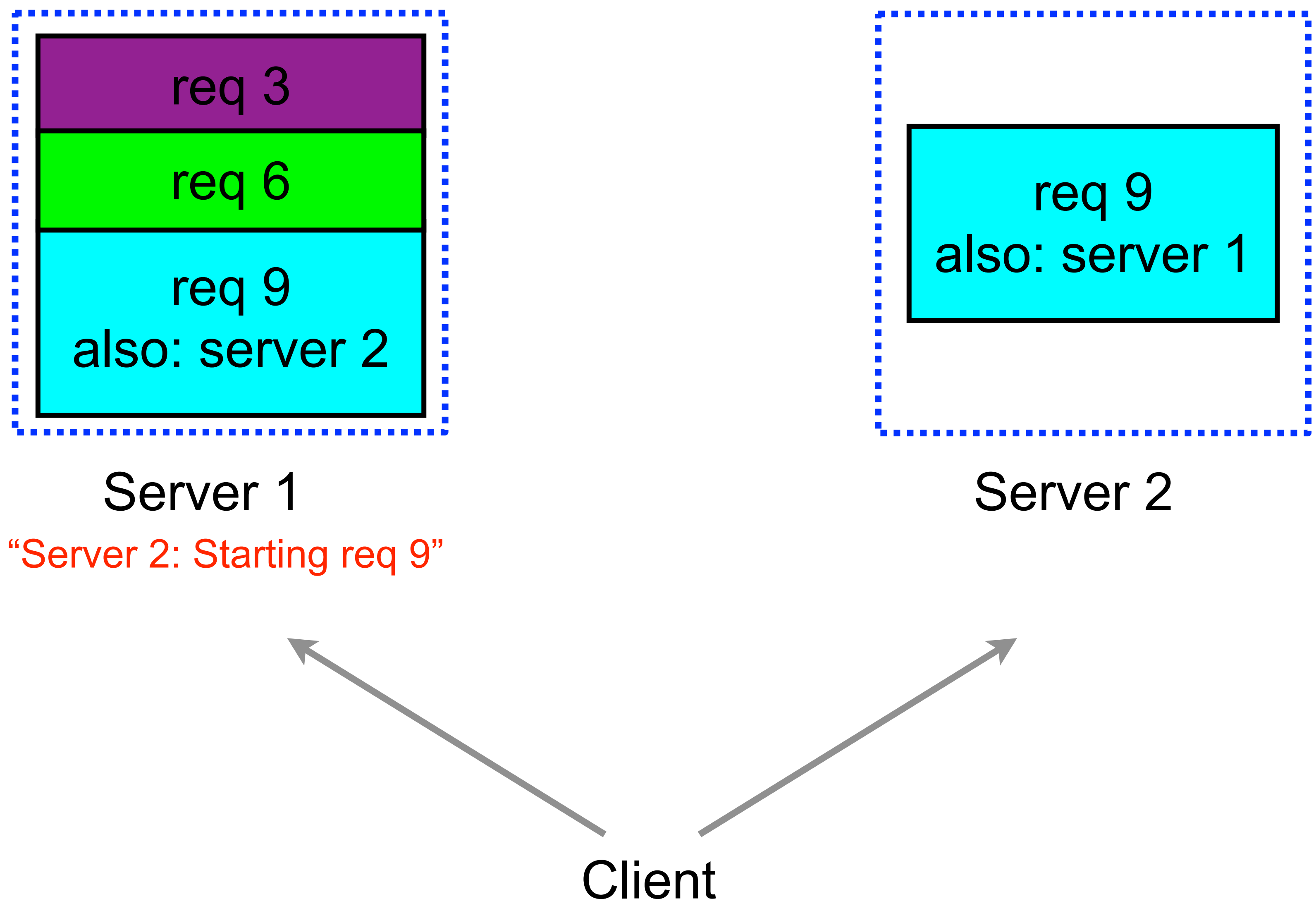
Tied Requests



Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one

Each request identifies other server(s) to which request might be sent

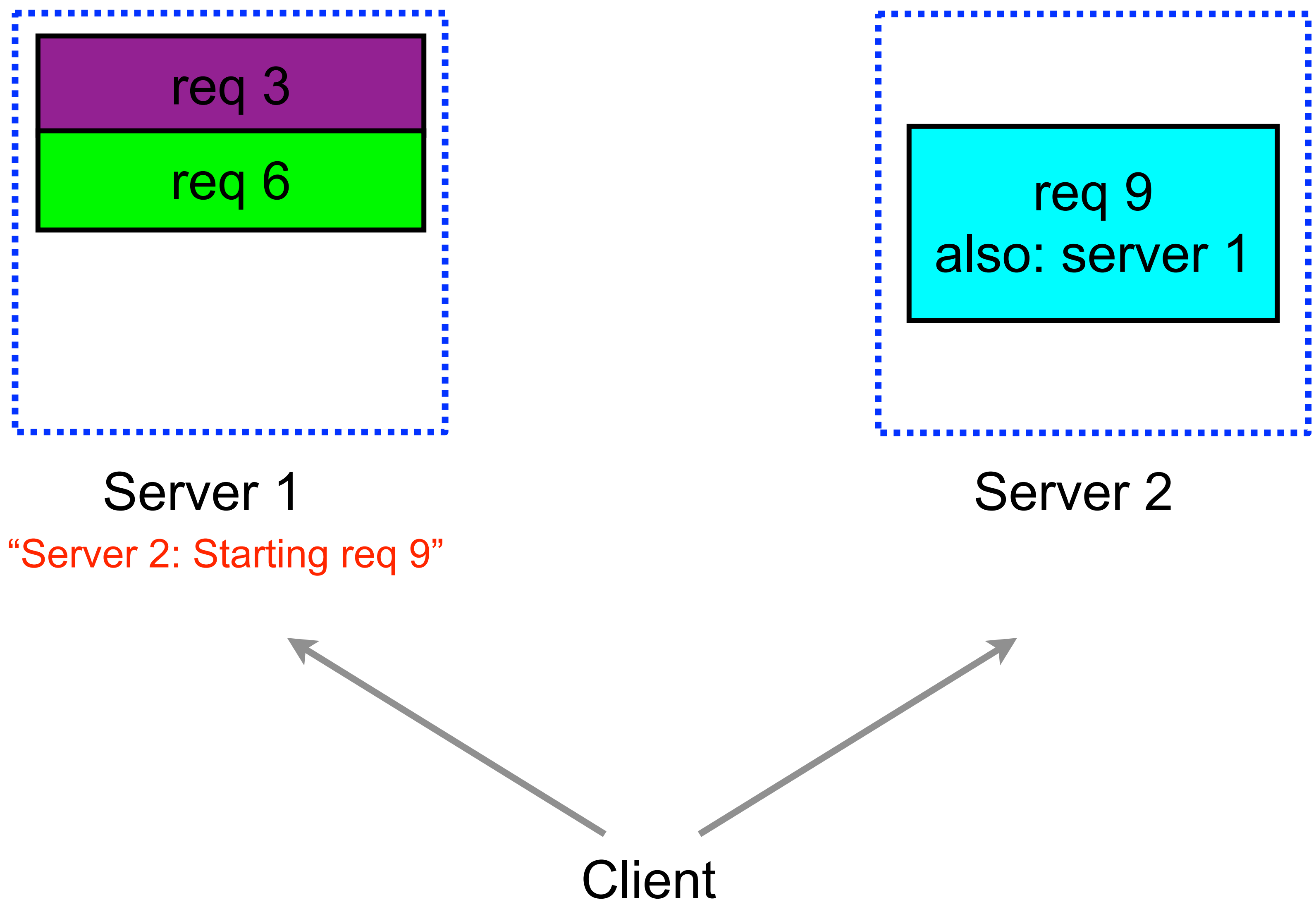
Tied Requests



Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one

Each request identifies other server(s) to which request might be sent

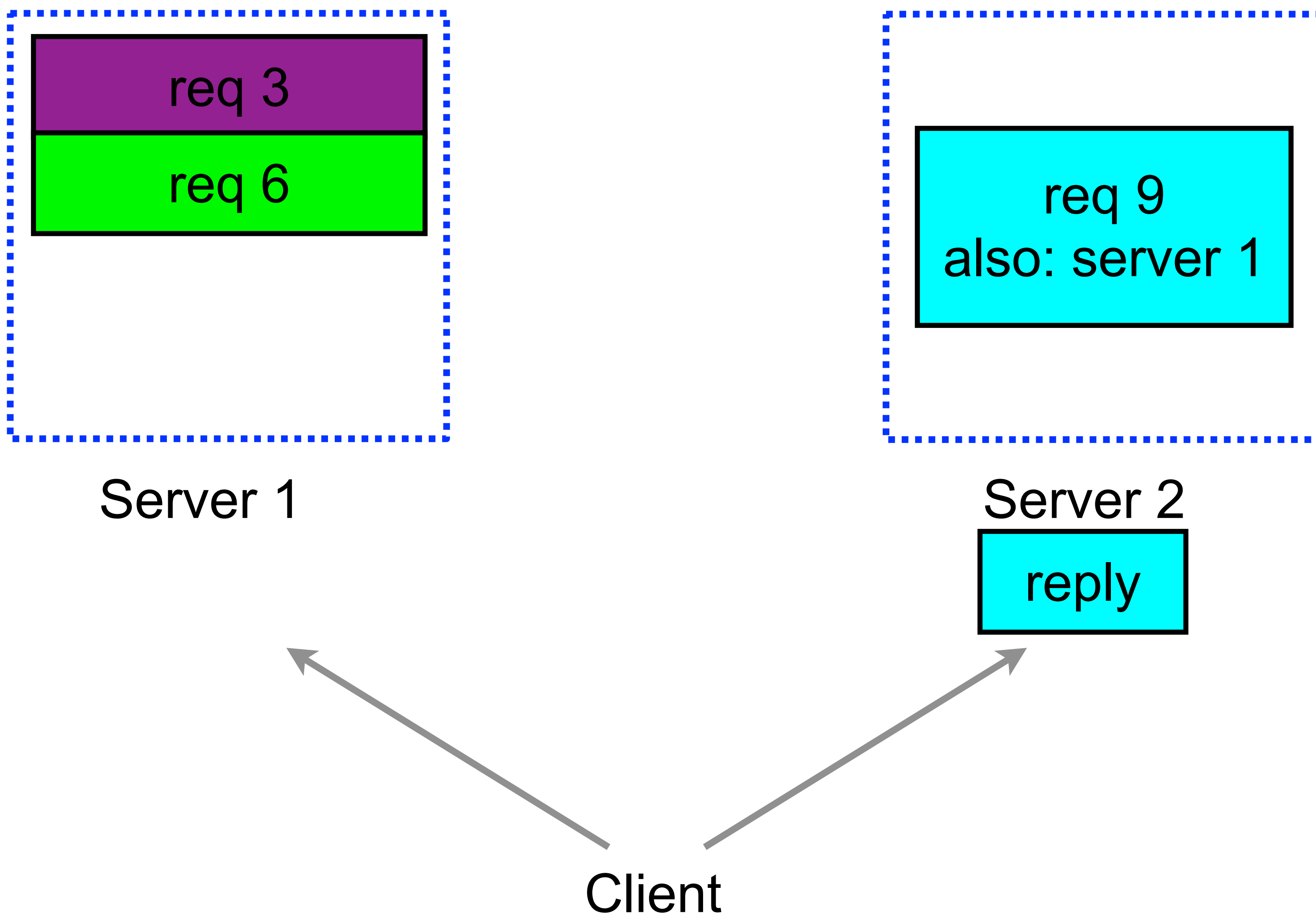
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent Google

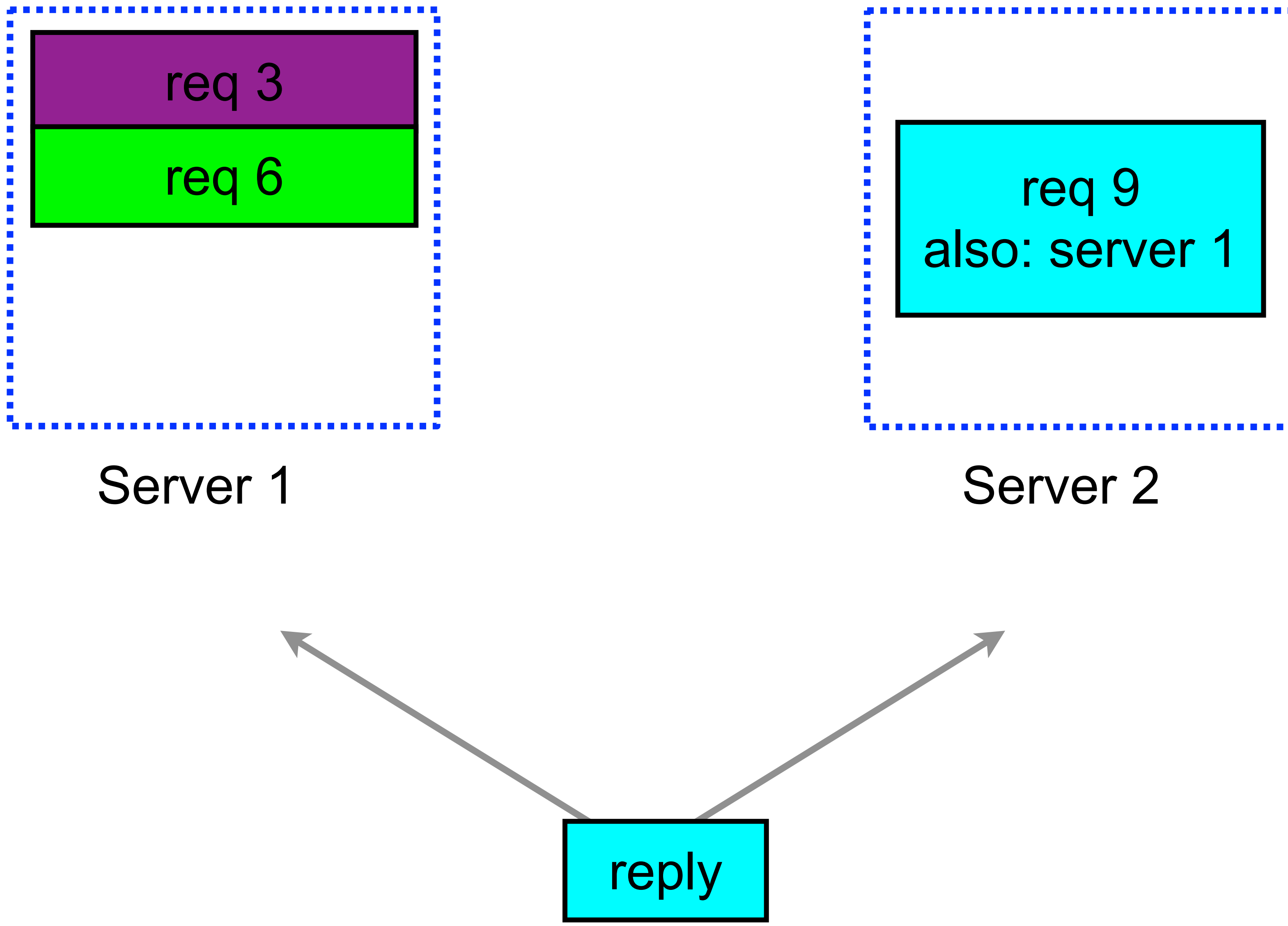
Tied Requests



Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent

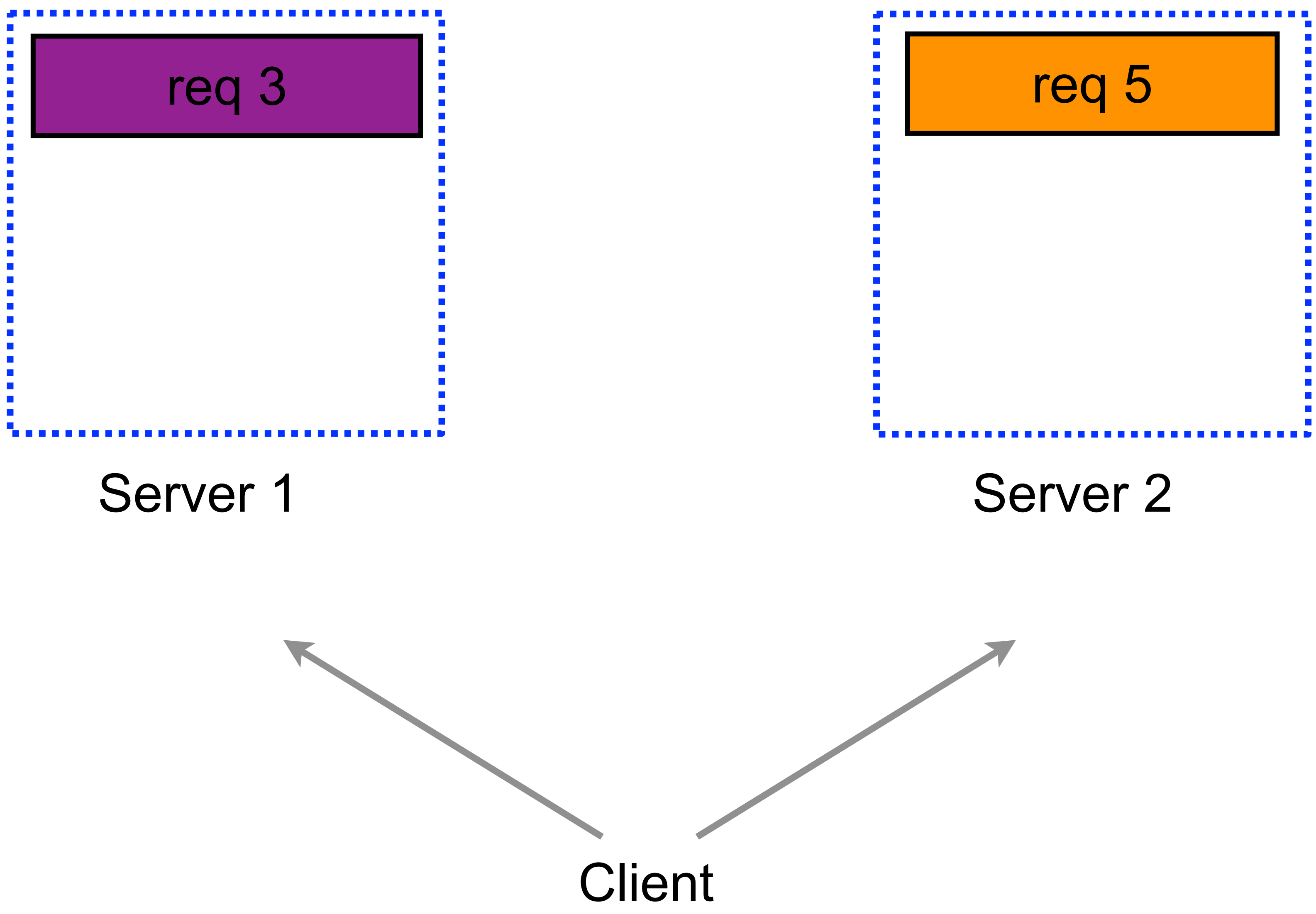
Tied Requests



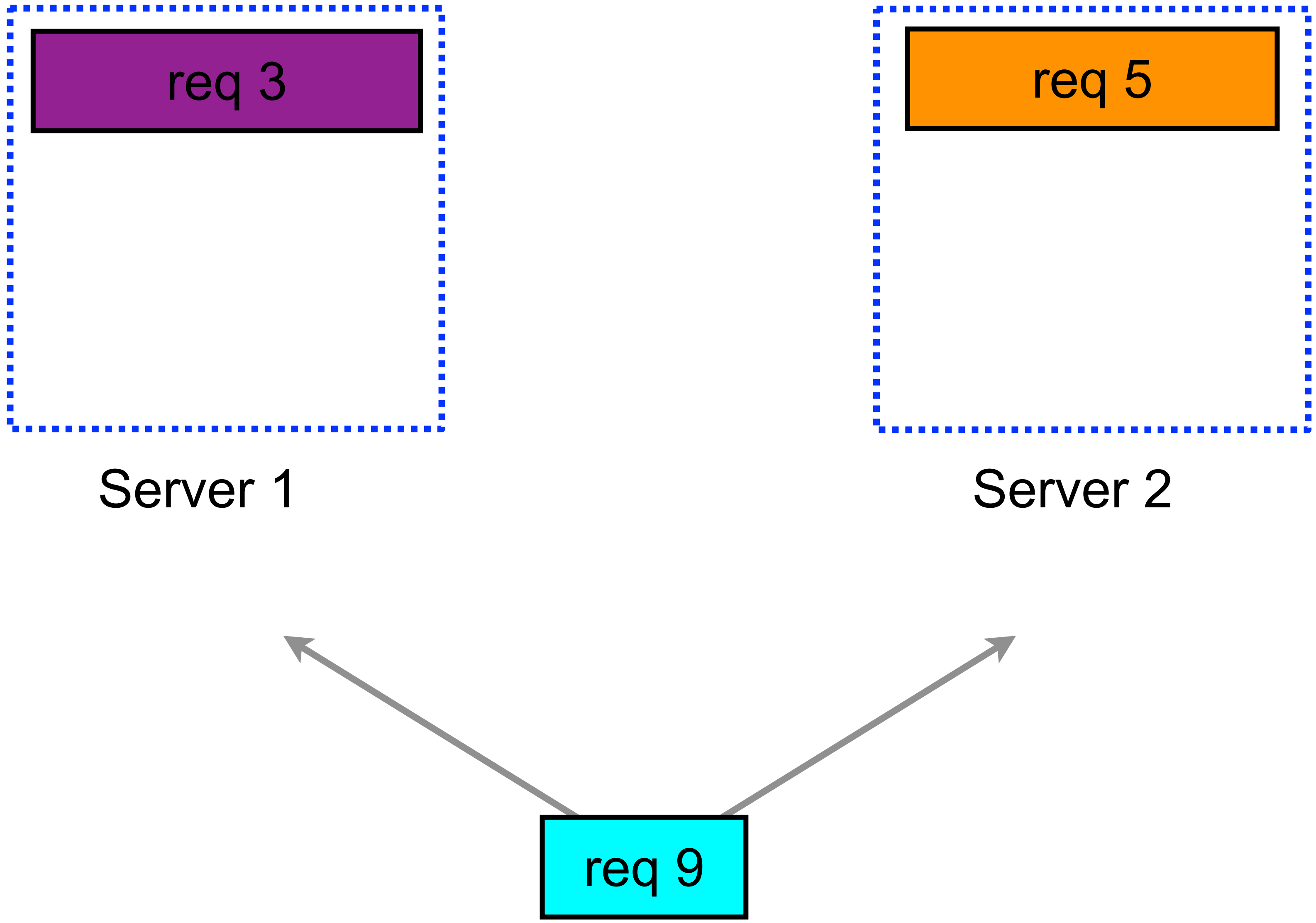
Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one

Each request identifies other server(s) to which request might be sent

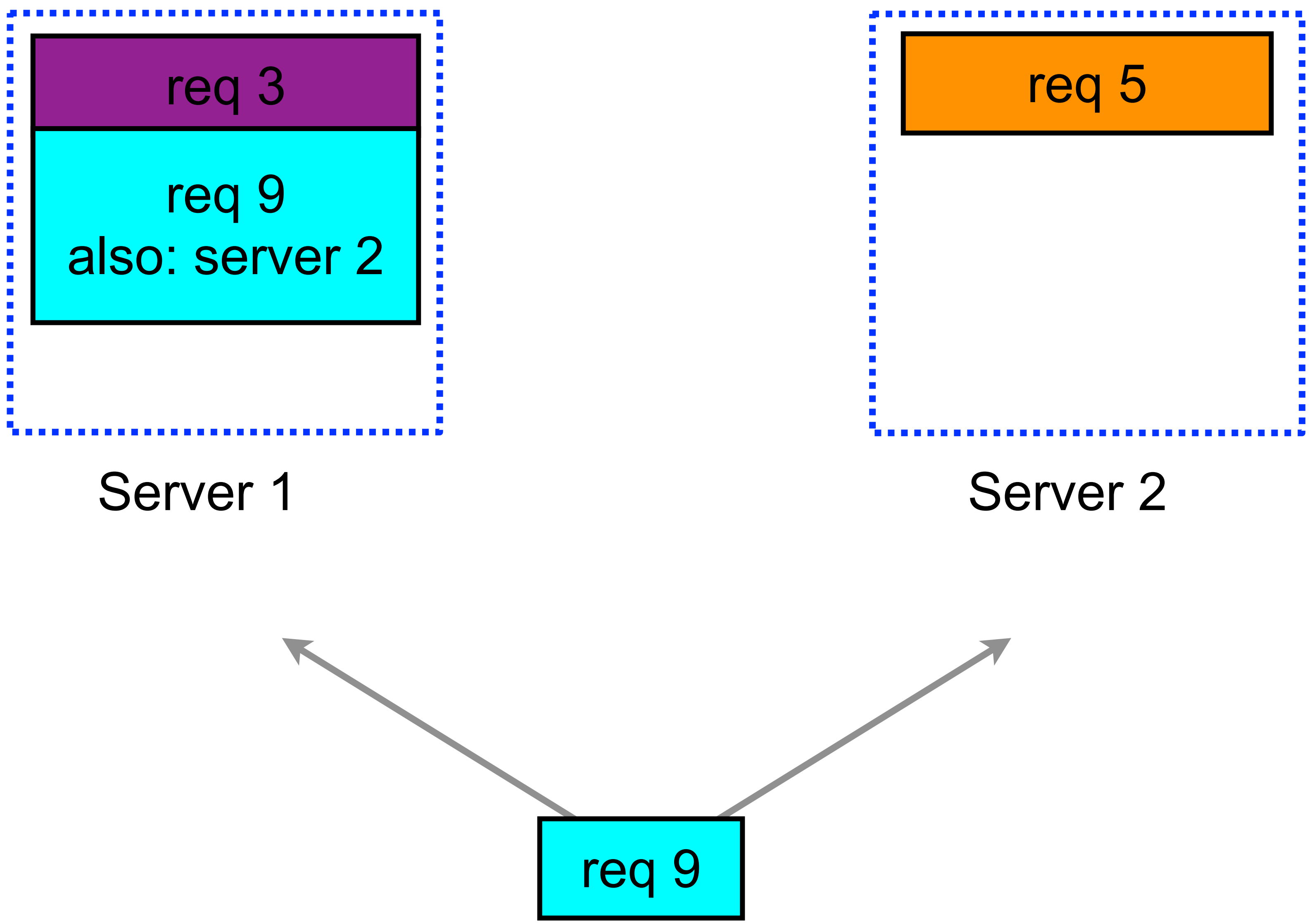
Tied Requests: Bad Case



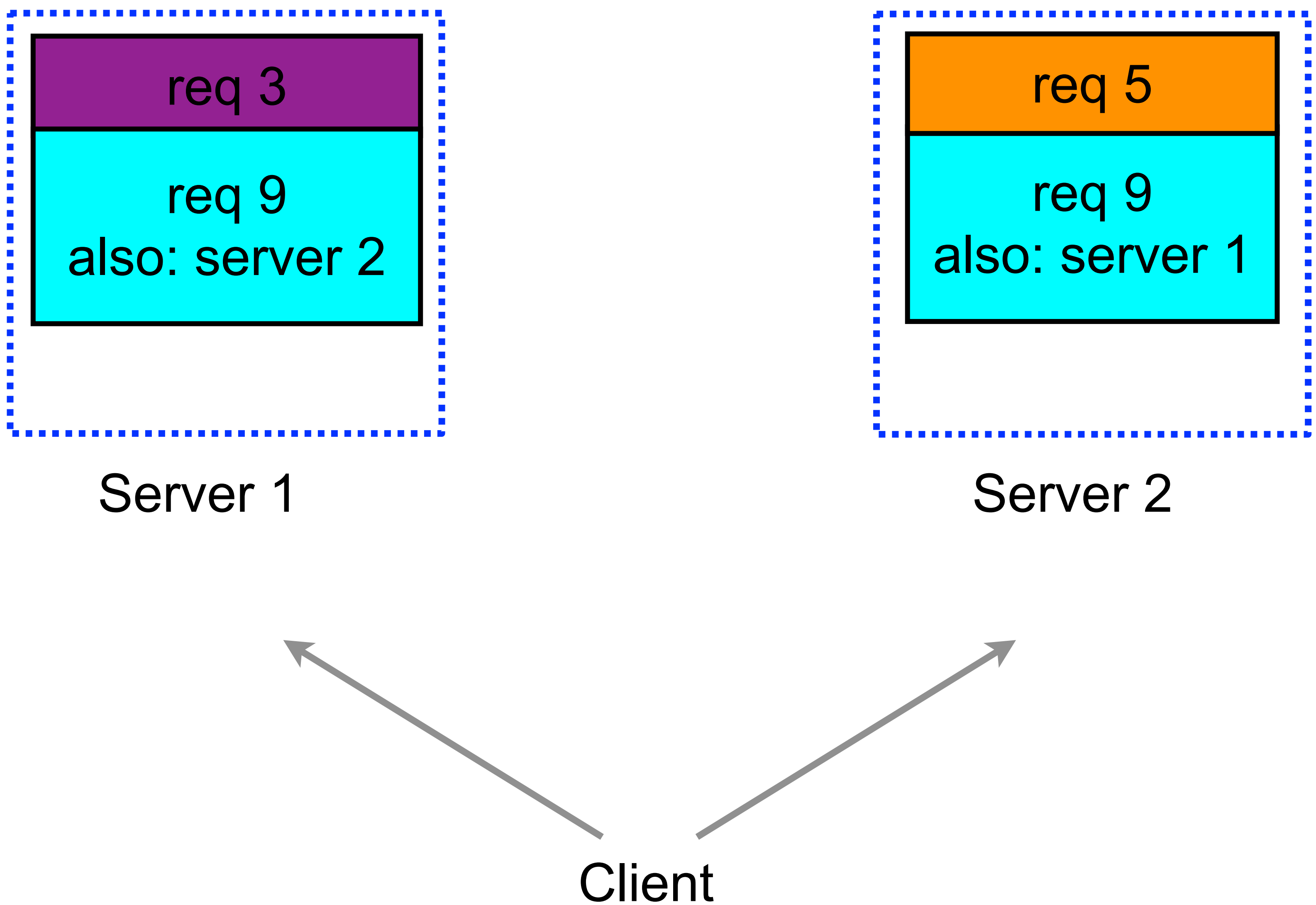
Tied Requests: Bad Case



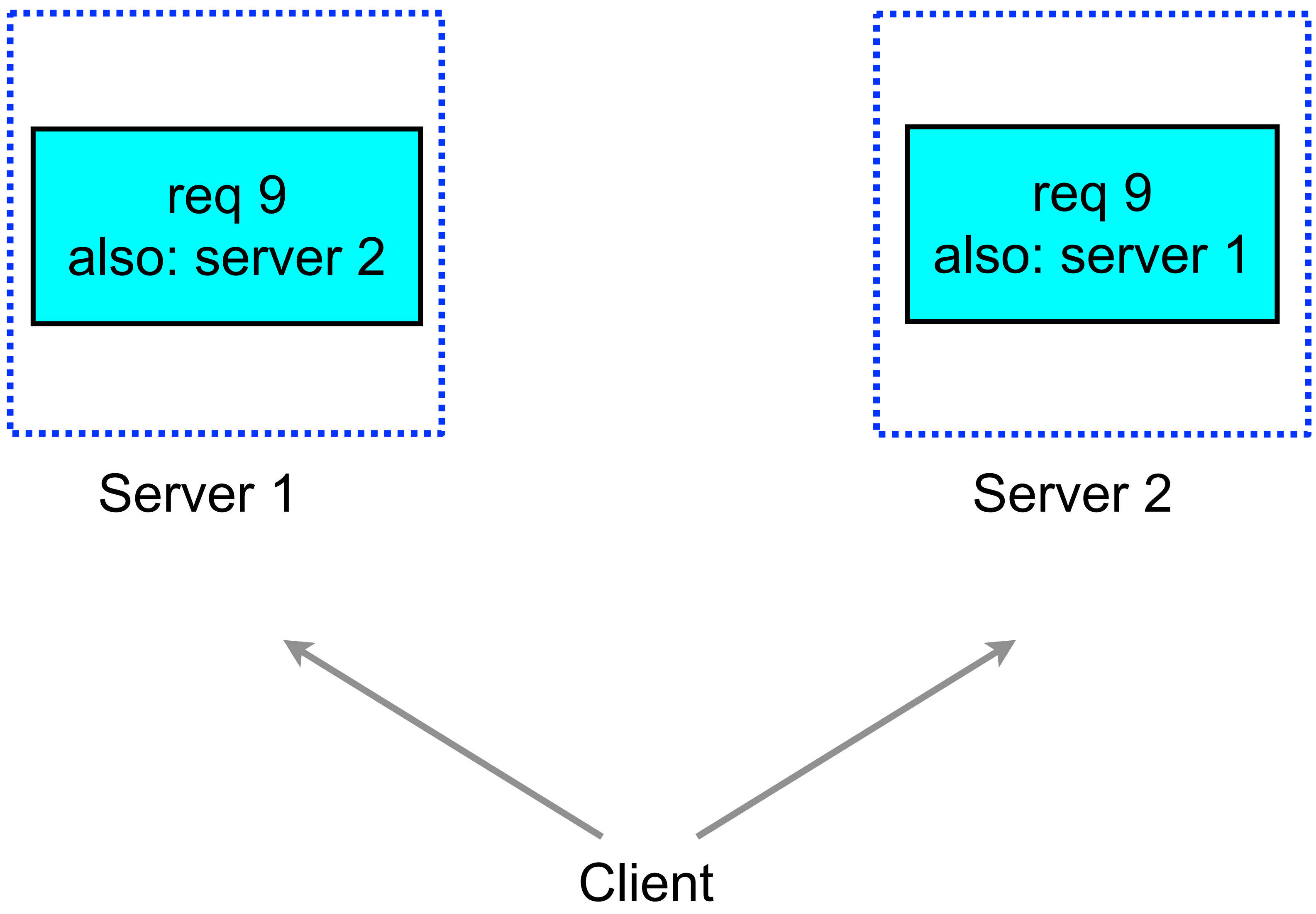
Tied Requests: Bad Case



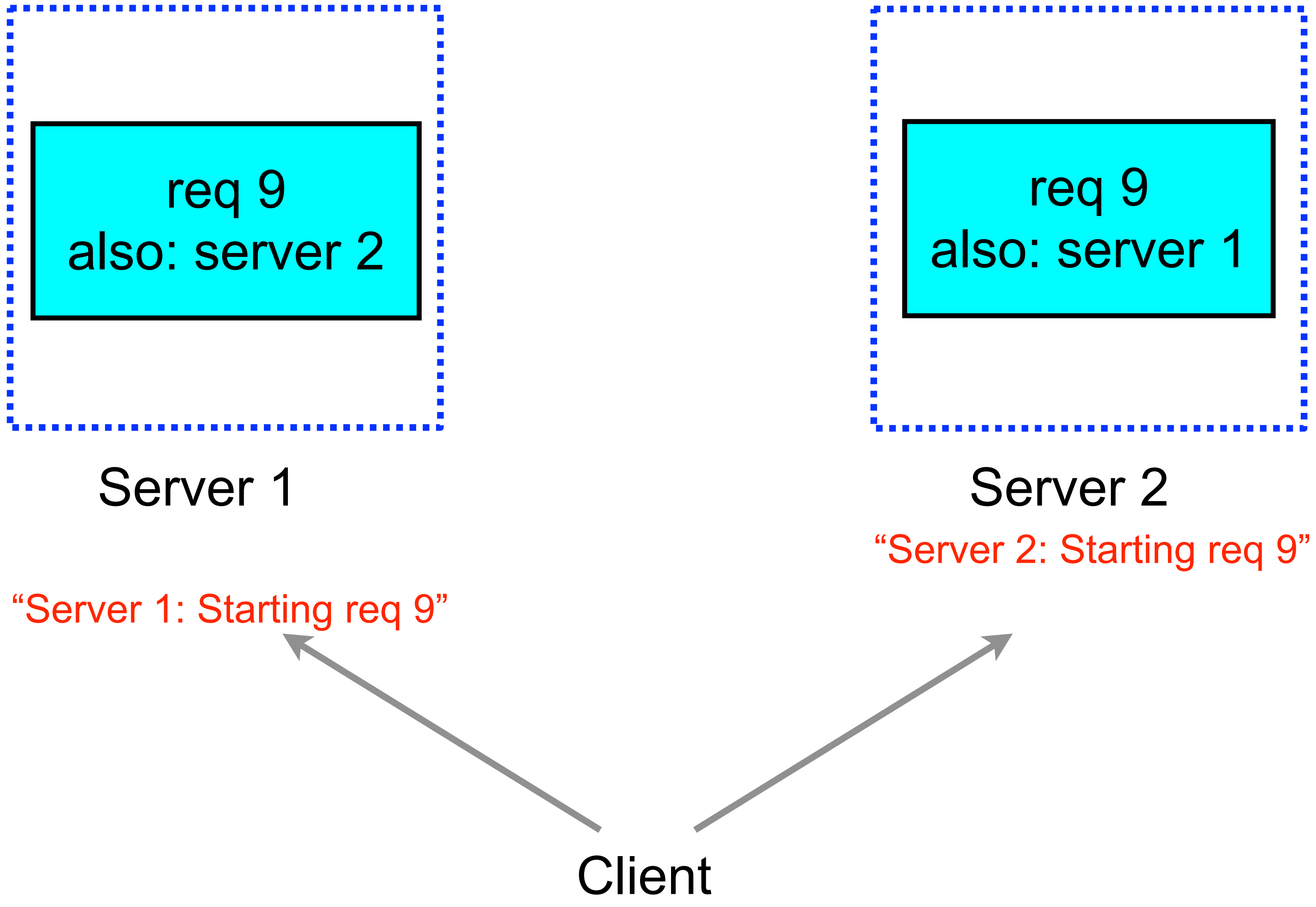
Tied Requests: Bad Case



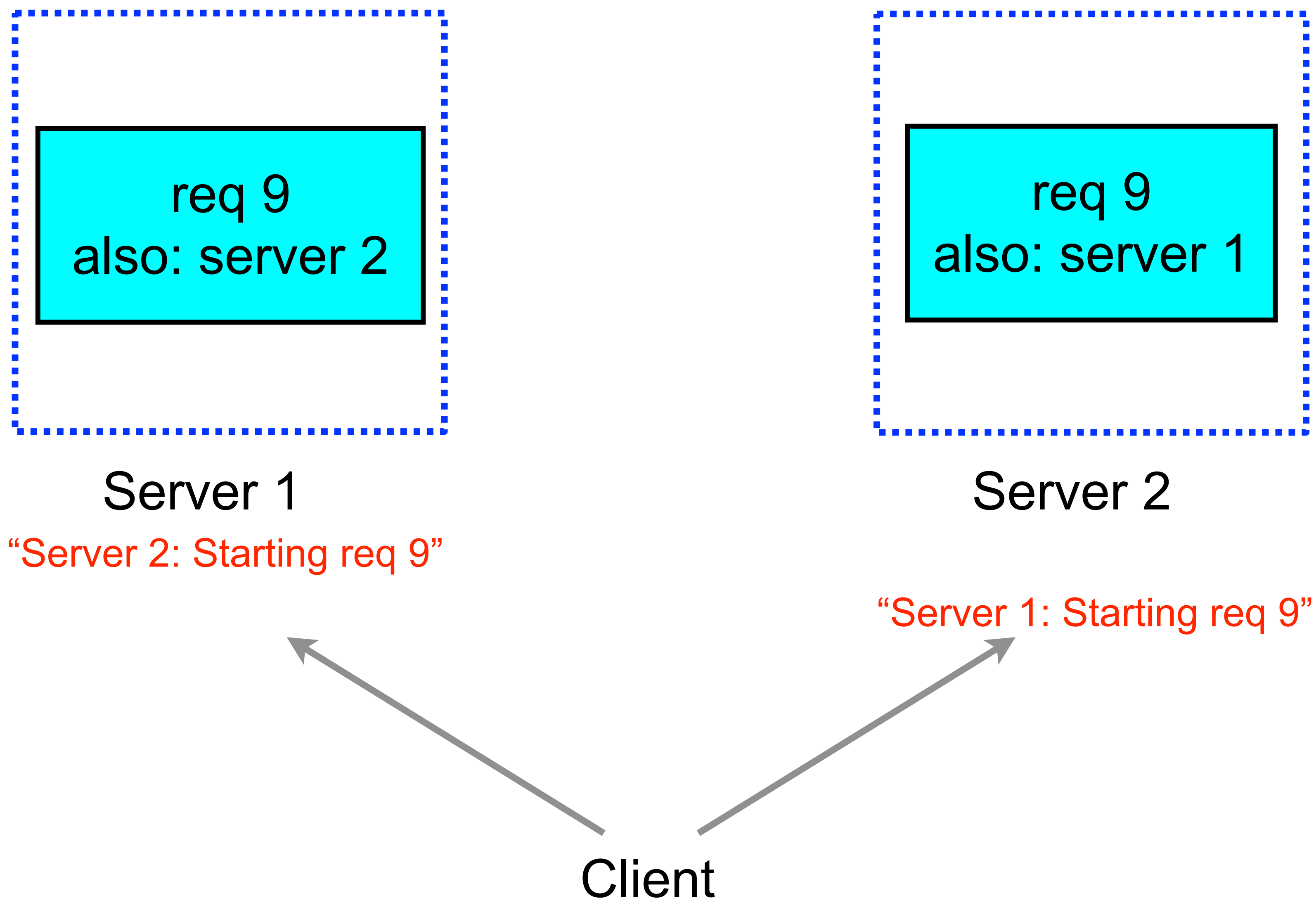
Tied Requests: Bad Case



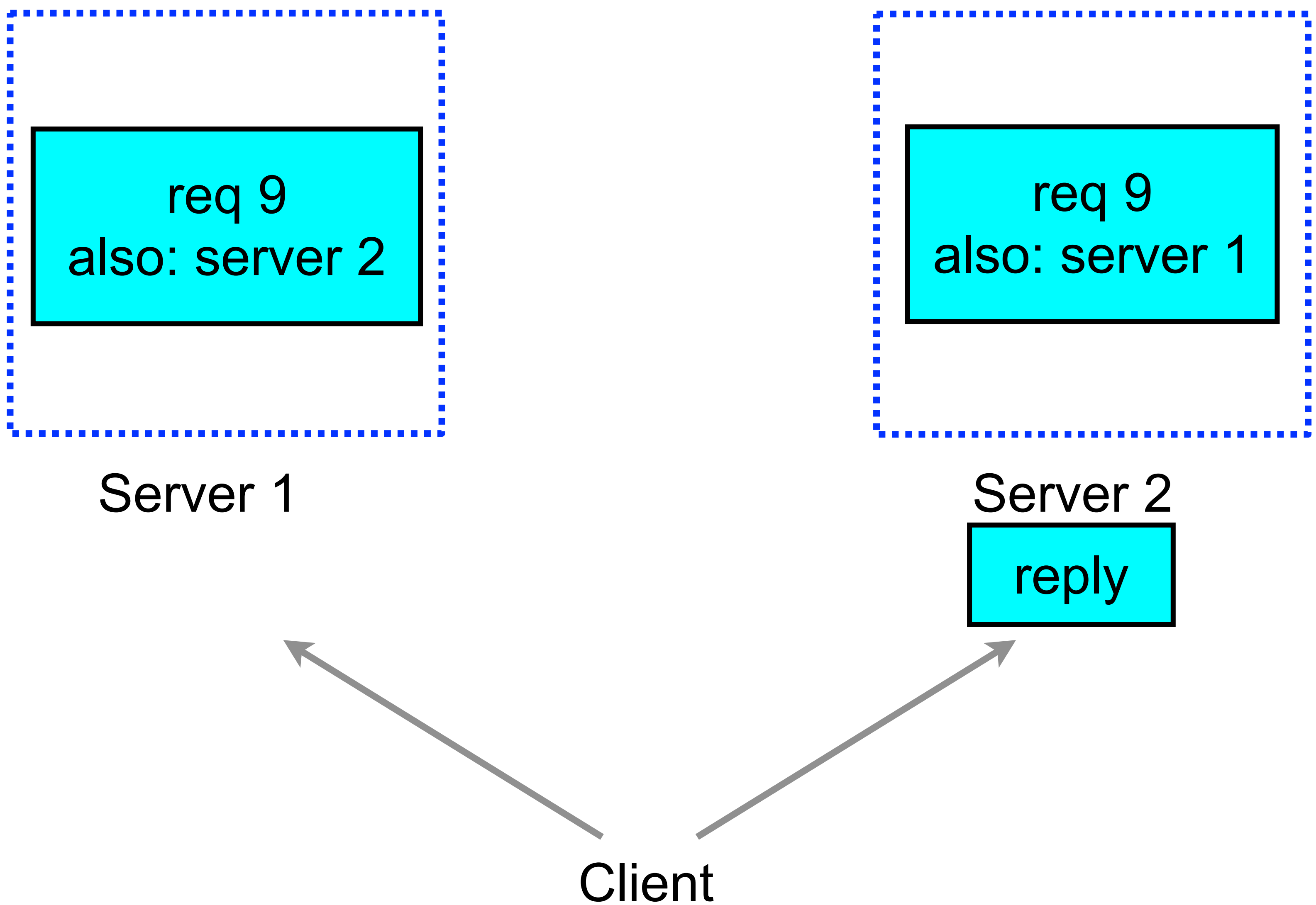
Tied Requests: Bad Case



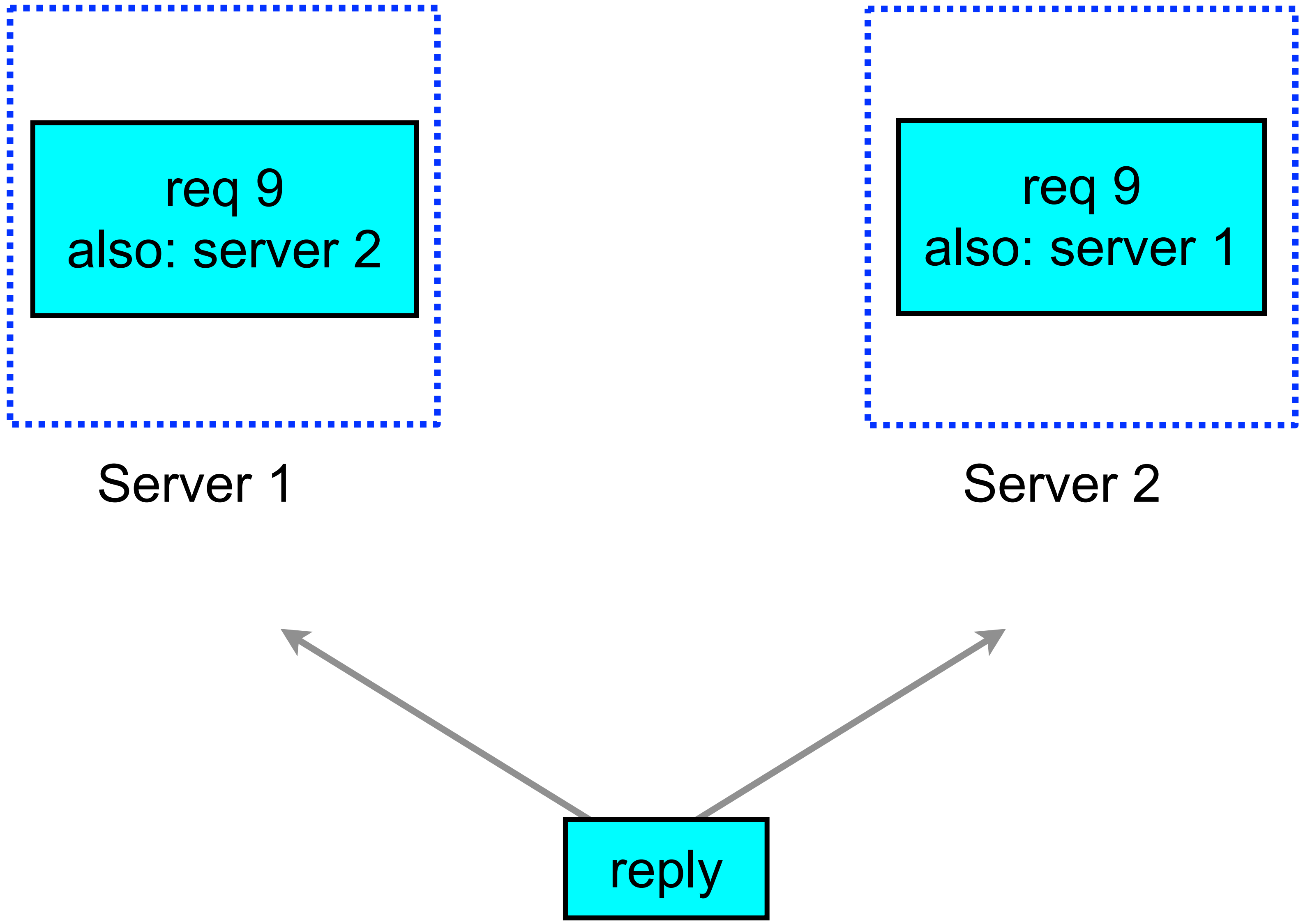
Tied Requests: Bad Case



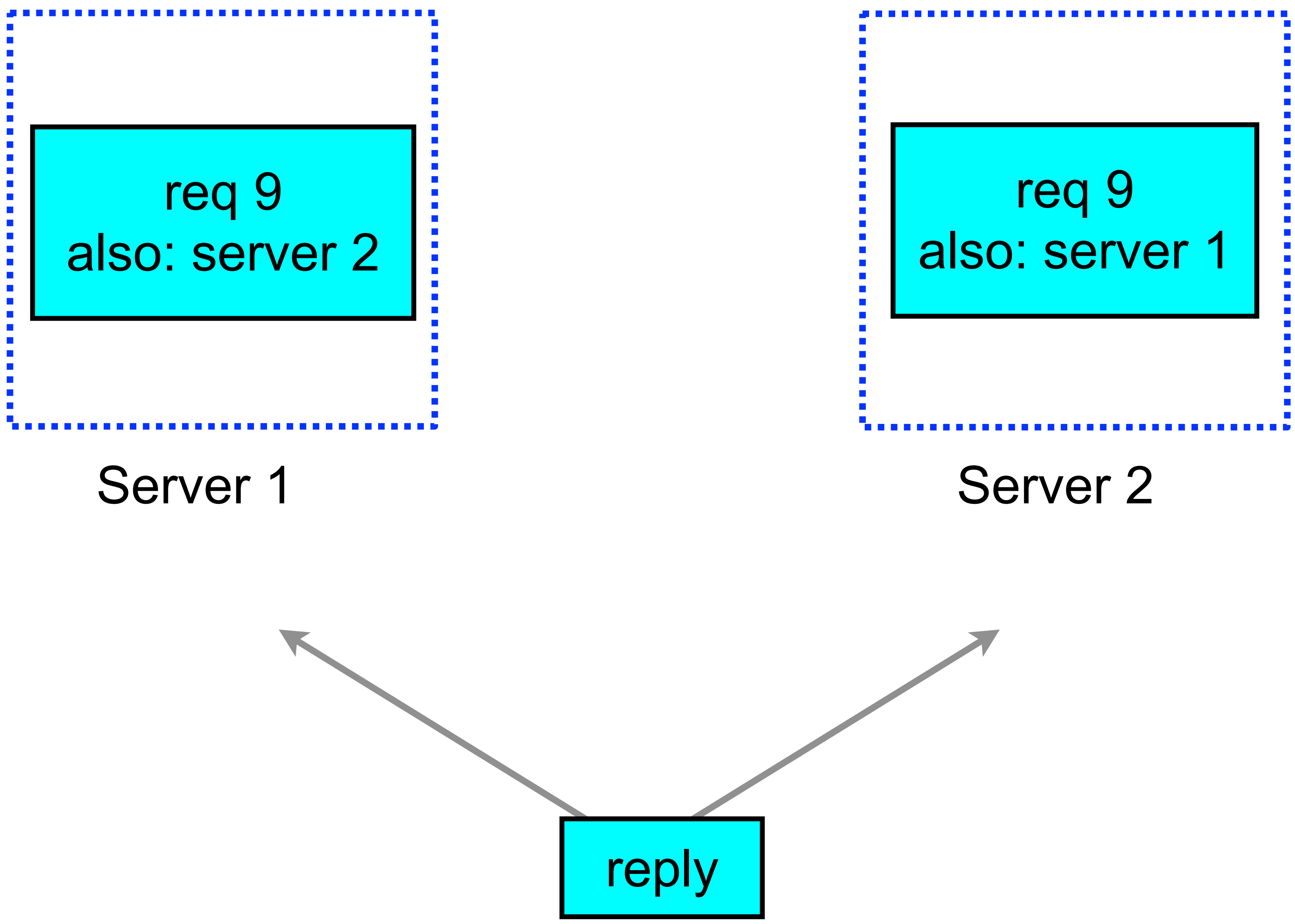
Tied Requests: Bad Case



Tied Requests: Bad Case



Tied Requests: Bad Case



Likelihood of this bad case is reduced with lower latency networks

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

-43%

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms

Google

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Google

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

-38%

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Google

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  – send tied request to first replica
  – wait 2 ms, and send tied request to second replica
  – servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Backups cause about ~1% extra disk reads

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Google

Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Backups w/big sort job gives same read latencies as no backups w/ idle cluster!

Google

Cluster-Level Services

- Our earliest systems made things easier within a cluster:
  - GFS/Colossus: reliable cluster-level file system
  - MapReduce: reliable large-scale computations
  - Cluster scheduling system: abstracted individual machines
  - BigTable: automatic scaling of higher-level structured storage

Cluster-Level Services

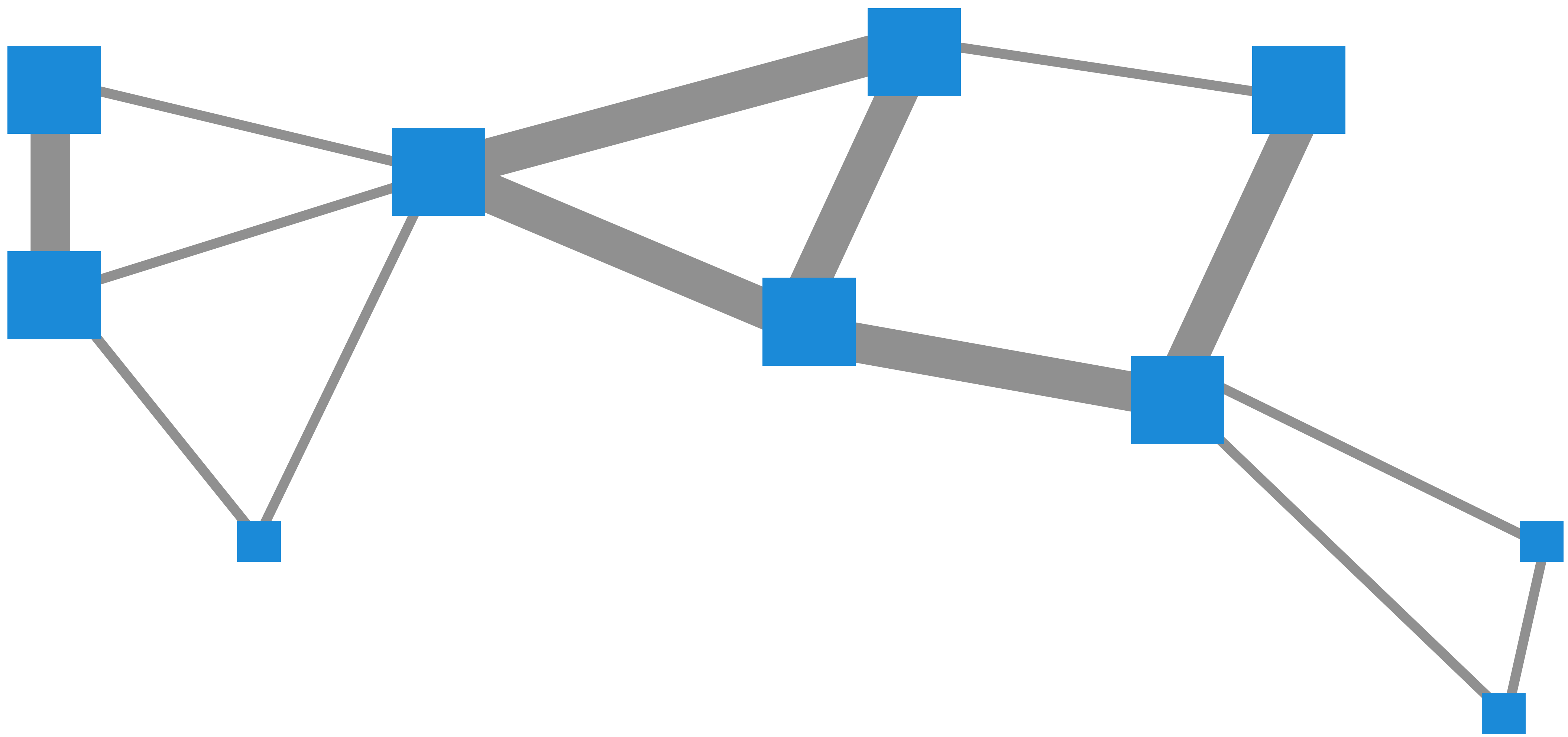
- Our earliest systems made things easier within a cluster:
  - GFS/Colossus: reliable cluster-level file system
  - MapReduce: reliable large-scale computations
  - Cluster scheduling system: abstracted individual machines
  - BigTable: automatic scaling of higher-level structured storage

- Solve many problems, but leave many cross-cluster issues to human-level operators

– different copies of same dataset have different names

– moving or deploying new service replicas is labor intensive

Spanner: Worldwide Storage



Google

Spanner: Worldwide Storage

- Single global namespace for data
- Consistent replication across datacenters
- Automatic migration to meet various constraints
  - resource constraints

    "The file system in this Belgian datacenter is getting full..."
  - application-level hints

    "Place this data in Europe and the U.S."

    "Place this data in flash, and place this other data on disk"

Google

Spanner: Worldwide Storage

- Single global namespace for data
- Consistent replication across datacenters
- Automatic migration to meet various constraints
  - resource constraints

    "The file system in this Belgian datacenter is getting full..."
  - application-level hints

    "Place this data in Europe and the U.S."

    "Place this data in flash, and place this other data on disk"

- System underlies Google's production advertising system, among other uses

- [*Spanner: Google's Globally-Distributed Database*, Corbett, Dean, ..., Ghemawat, ... et al., to appear in OSDI 2012]

Monitoring and Debugging

- Questions you might want to ask:
  - did this change I rolled out last week affect # of errors / request?
  - why are my tasks using so much memory?
  - where is CPU time being spent in my application?
  - what kinds of requests are being handled by my service?
  - why are some requests very slow?

- Important to have enough visibility into systems to answer these kinds of questions

Exported Variables

- Special URL on every Google server

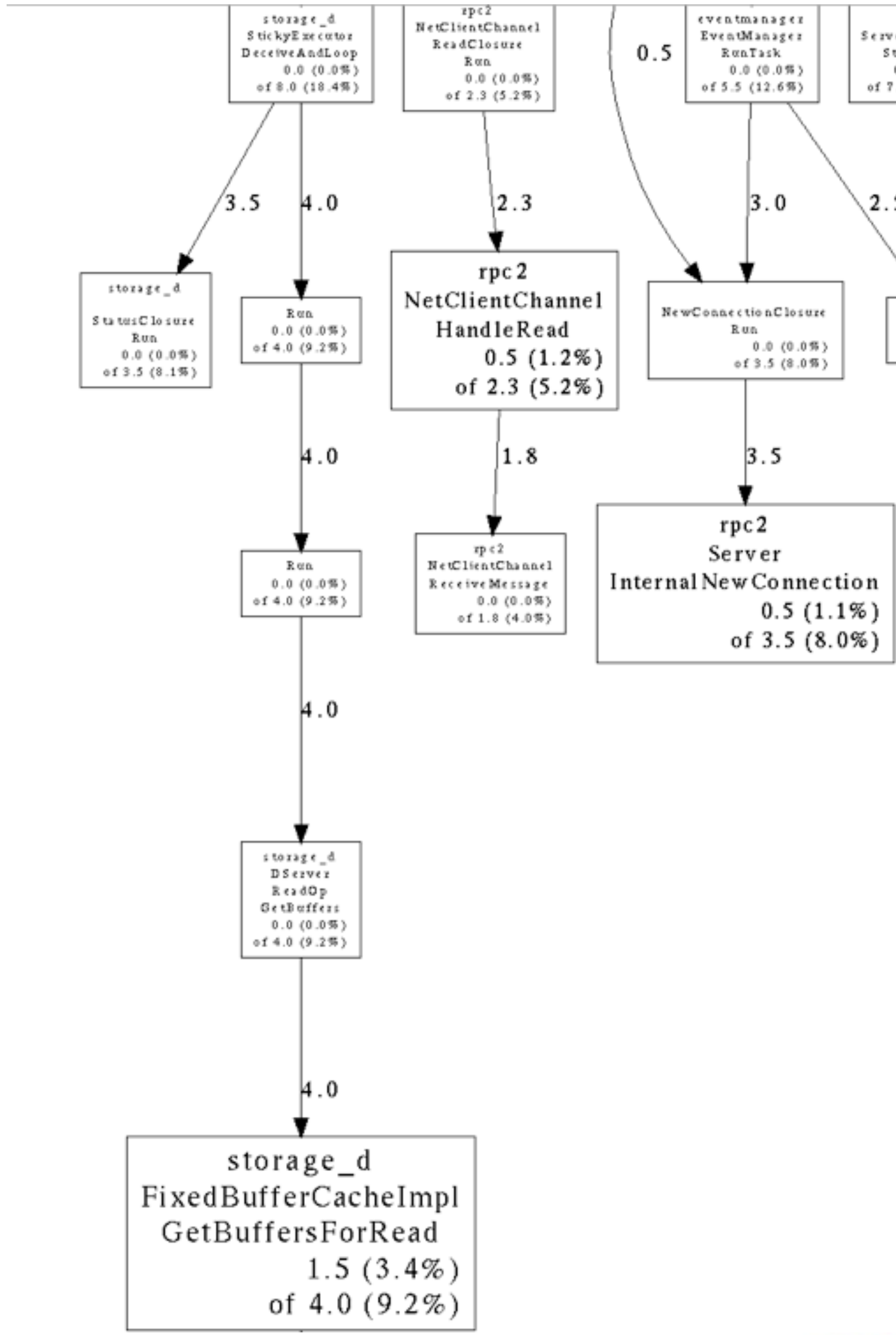
```
rpc-server-count-minute                    11412
rpc-server-count                       502450983
rpc-server-arg-bytes-minute              8039419
rpc-server-arg-bytes                372908296166
rpc-server-rpc-errors-minute                   0
rpc-server-rpc-errors                          0
rpc-server-app-errors-minute                   8
rpc-server-app-errors                    2357783
uptime-in-ms                           679532636
build-timestamp-as-int                1343415737
build-timestamp "Built on Jul 27 2012 12:02:17 (1343415737)"
...
```

- On top of this, we have systems that gather all of this data
  - can aggregate across servers & services, compute derived values, graph data, examine historical changes, etc.

Online Profiling

- Every server supports sampling-based hierarchical profiling
  - CPU
  - memory usage
  - lock contention time
- Example: memory sampling
  - every Nth byte allocated, record stack trace of where allocation occurred
  - when sampled allocation is freed, drop stack trace
  - (N is large enough that overhead is small)

Memory Profile



Request Tracing

- Every client and server gathers sample of requests
  - different sampling buckets, based on request latency

```
   2012/09/09-11:39:21.029630       0.018978 Read (trace_id: c6143c073204f13f ...)
11:39:21.029611      -0.000019 ... RPC: 07eb70184bfff86f ... deadline:0.8526s
11:39:21.029611      -0.000019 ... header:<path:"..." length:33082 offset:3037807
11:39:21.029729        .    99 ... StartRead(..., 3037807, 33082)
11:39:21.029730        .     1 ... ContentLock
11:39:21.029732        .     2 ... GotContentLock
...
11:39:21.029916        .     2 ... IssueRead
11:39:21.048196        . 18280 ... HandleRead: OK
11:39:21.048666        .   431 ... RPC: OK [33082 bytes]
```

Request Tracing

- Every client and server gathers sample of requests
  – different sampling buckets, based on request latency

```
   2012/09/09-11:39:21.029630       0.018978 Read (trace_id: c6143c073204f13f ...)
11:39:21.029611      -0.000019 ... RPC: 07eb70184bfff86f ... deadline:0.8526s
11:39:21.029611      -0.000019 ... header:<path:"..." length:33082 offset:3037807
11:39:21.029729          .    99 ... StartRead(..., 3037807, 33082)
11:39:21.029730          .     1 ... ContentLock
11:39:21.029732          .     2 ... GotContentLock
...
11:39:21.029916          .     2 ... IssueRead
11:39:21.048196          . 18280 ... HandleRead: OK
11:39:21.048666          .   431 ... RPC: OK [33082 bytes]
```

- Dapper: cross-machine view of preceding information
  – can understand complex behavior across many services
  – [*Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*, Sigelman et al., 2010]

Google

Higher Level Systems

- Systems that provide high level of abstraction that “just works” are incredibly valuable:
  - GFS, MapReduce, BigTable, Spanner, transparent latency reduction techniques, etc.

- Can we build high-level systems that just work in other domains like machine learning?

Scaling Deep Learning

- Much of Google is working on approximating AI. AI is hard
  - Many people at Google spend countless person-years hand-engineering complex features to feed as input to machine learning algorithms

- Is there a better way?

- Deep Learning: Use very large scale brain simulations
  - improve many Google applications
  - make significant advances towards perceptual AI

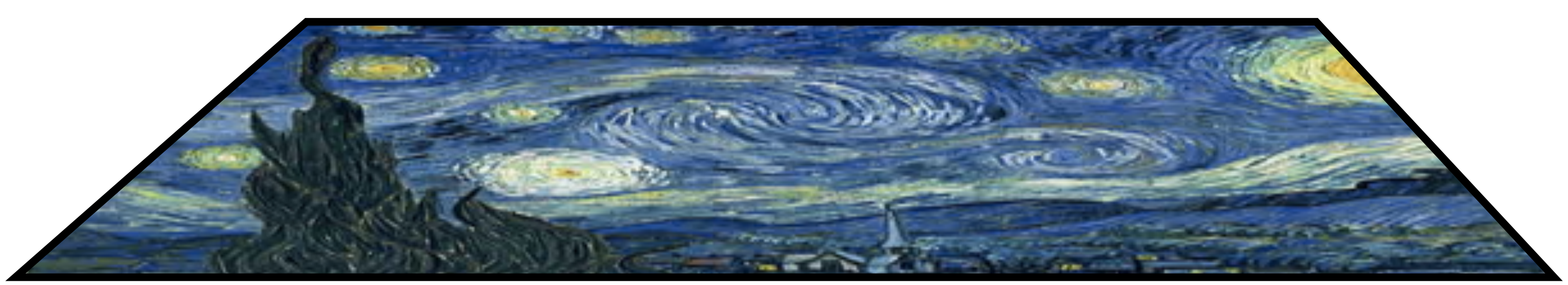
Deep Learning

- Algorithmic approach
  - automatically learn high-level representations from raw data
  - can learn from both labeled and unlabeled data

- Recent academic deep learning results improve on state-of-the-art in many areas:
  - images, video, speech, NLP, ...
  - ... using modest model sizes (<= ~50M parameters)

- We want to scale this approach up to much bigger models
  - currently: ~2B parameters, want ~10B-100B parameters
  - general approach: parallelize at many levels

Deep Networks

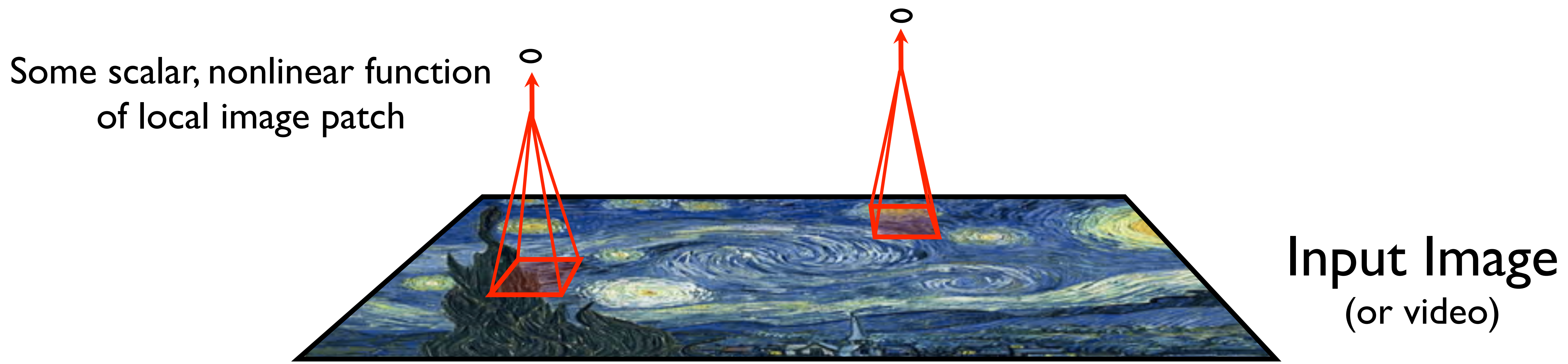


Input Image
(or video)

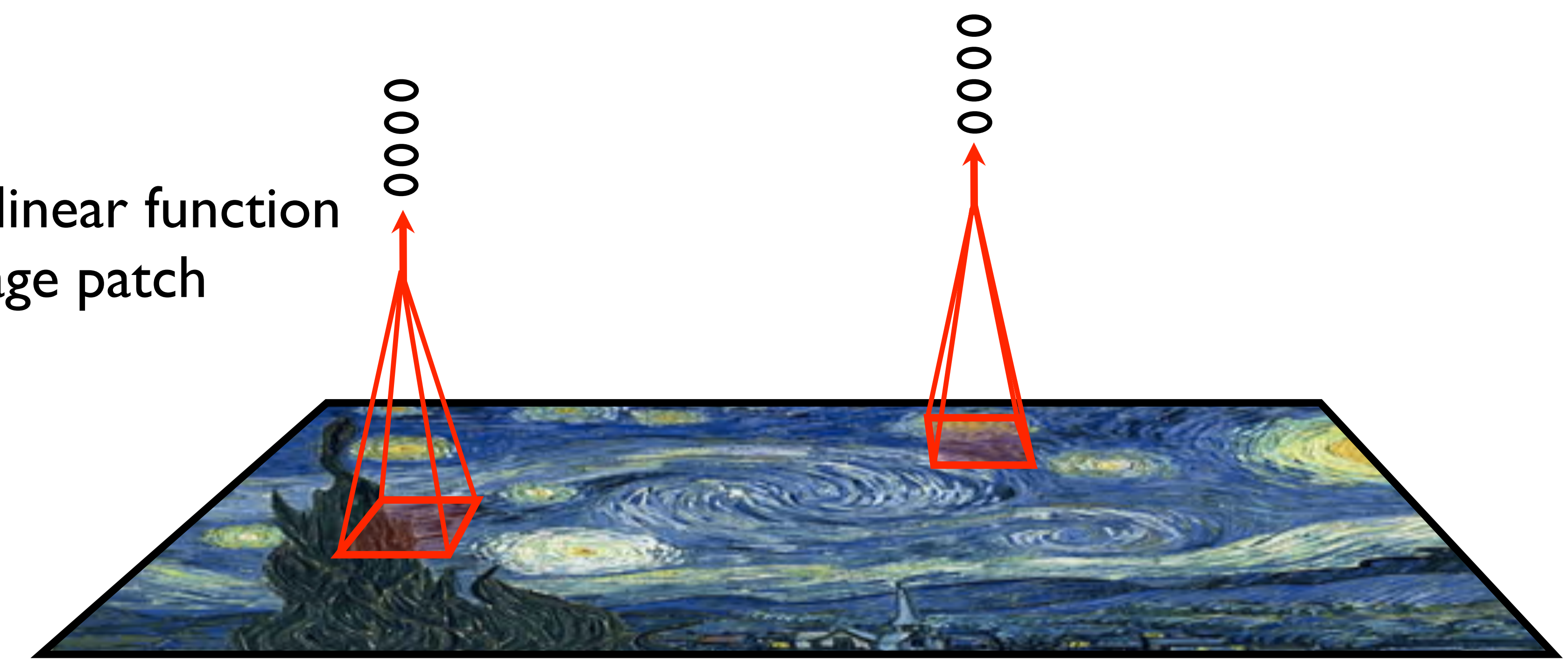
Deep Networks

Deep Networks

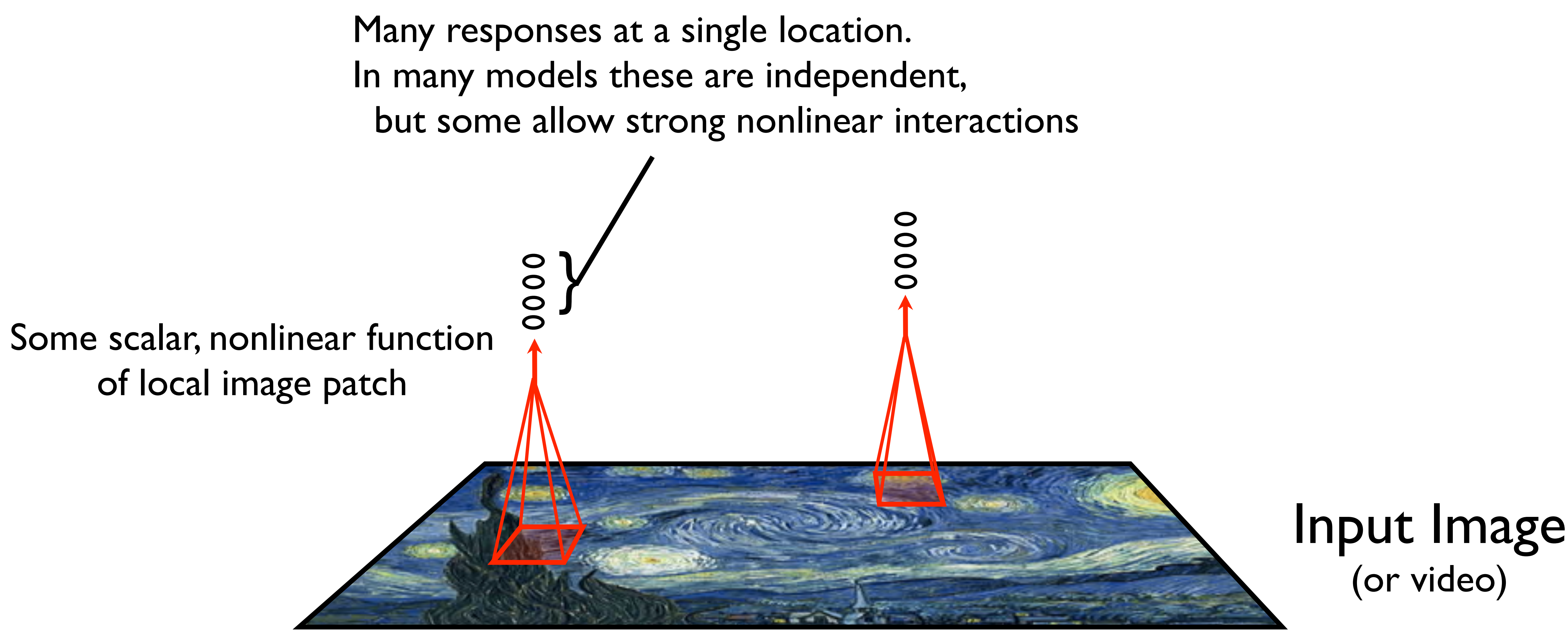
Deep Networks



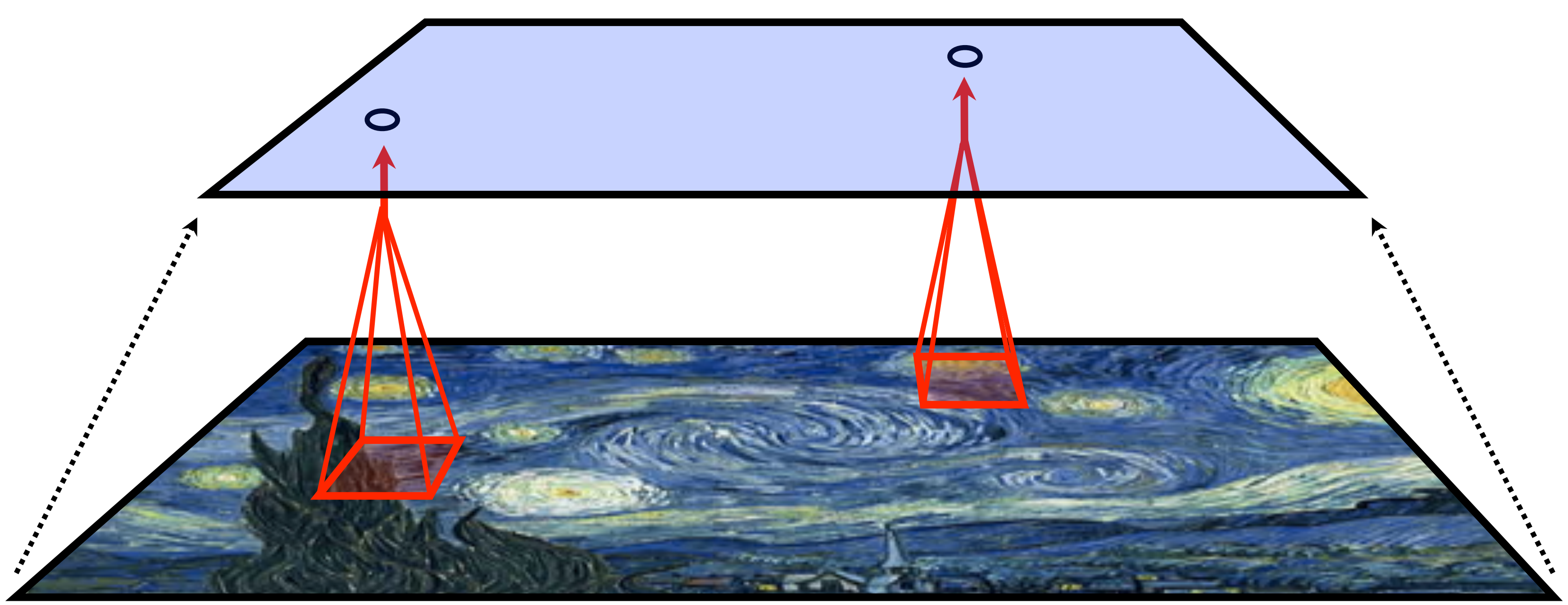
Deep Networks



Deep Networks

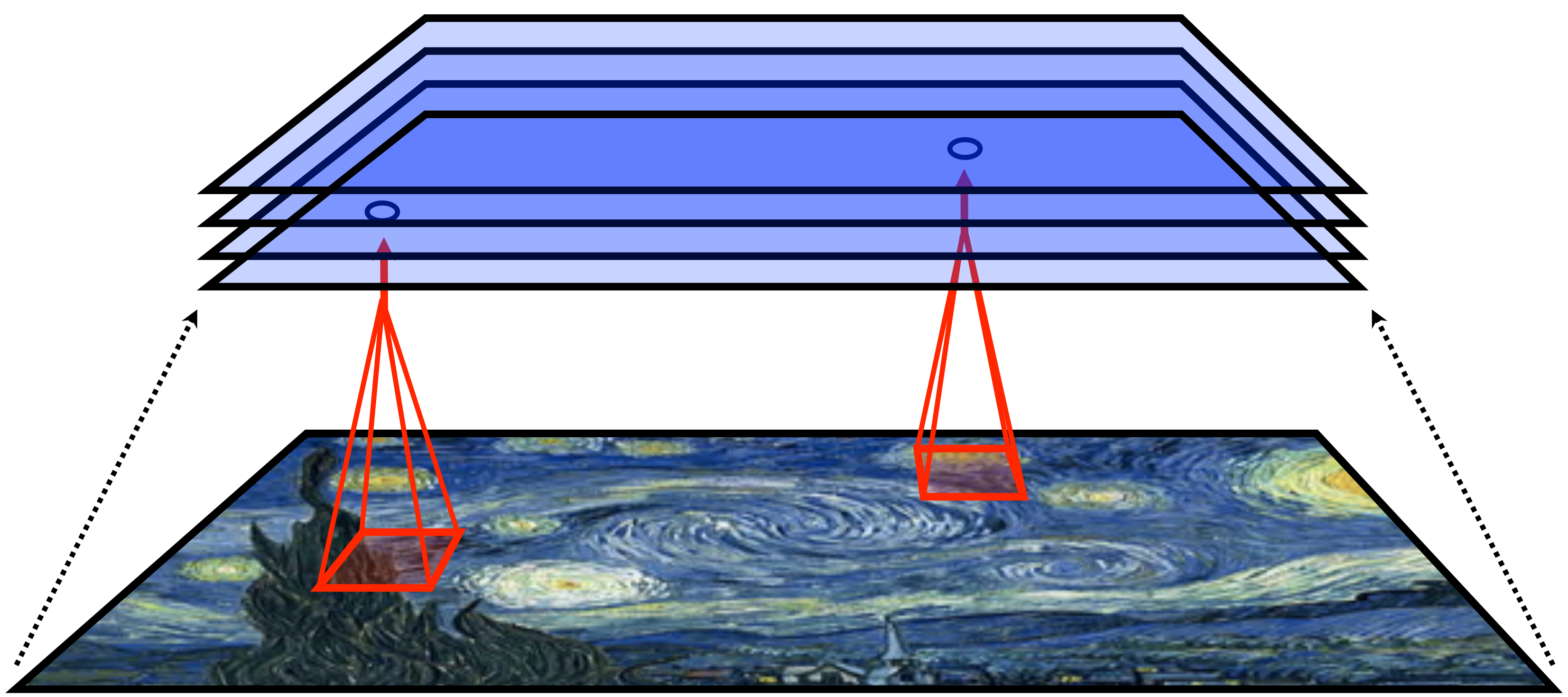


Deep Networks

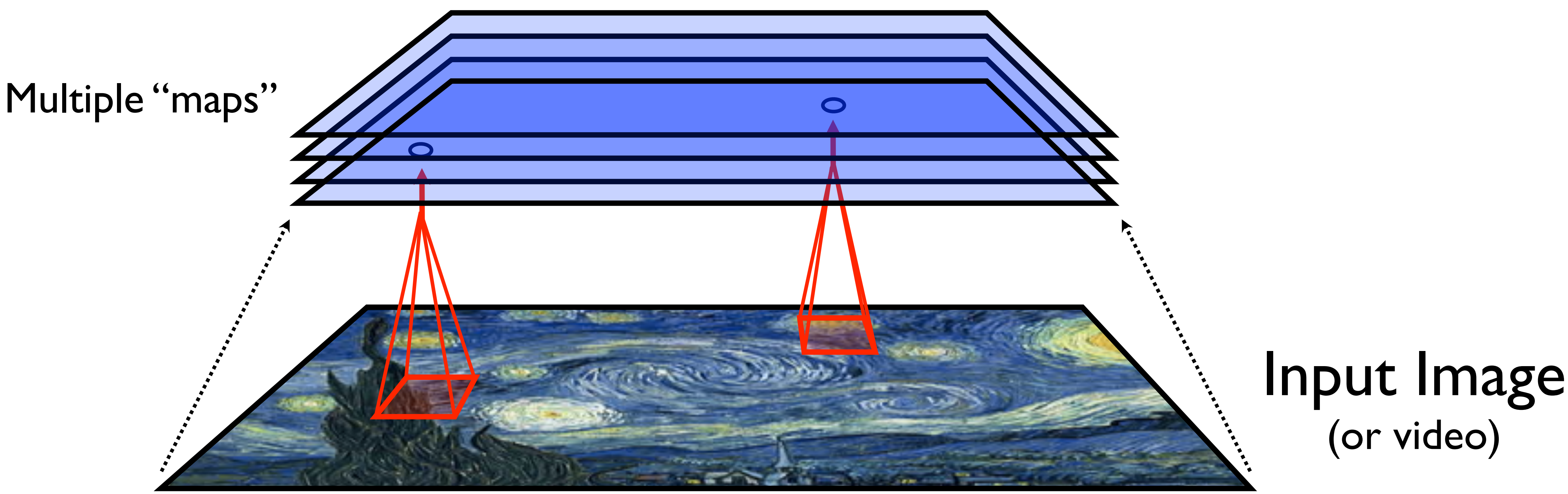


Input Image
(or video)

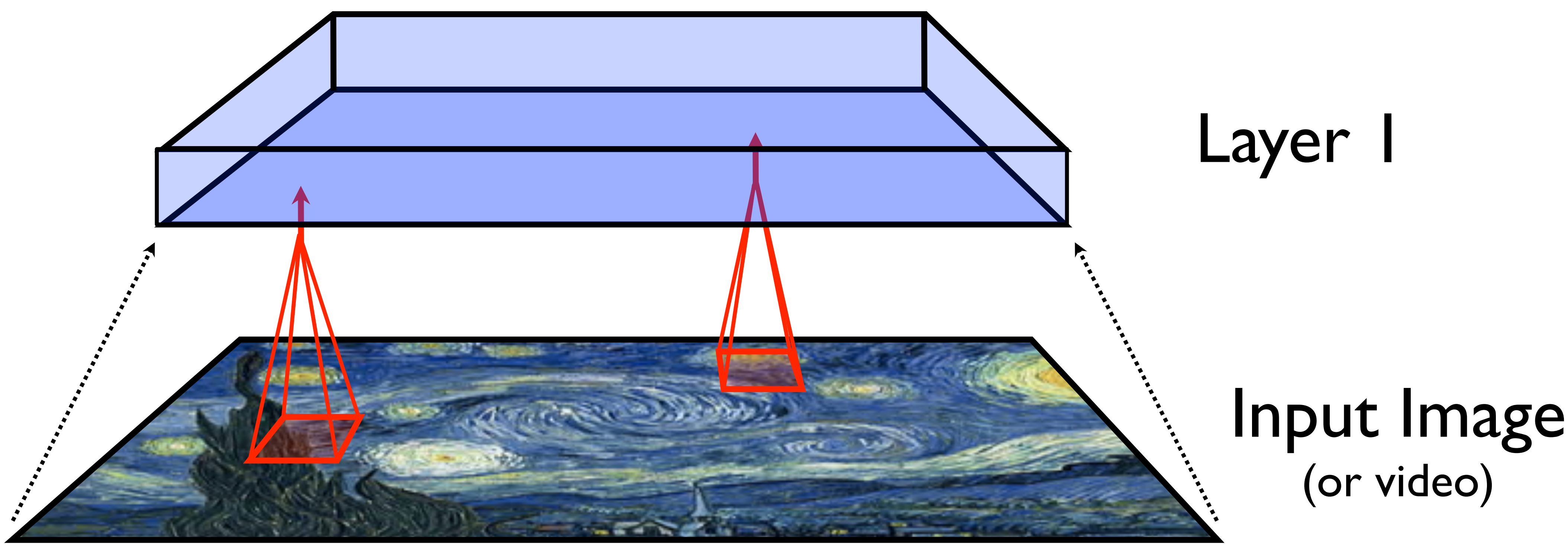
Deep Networks



Deep Networks

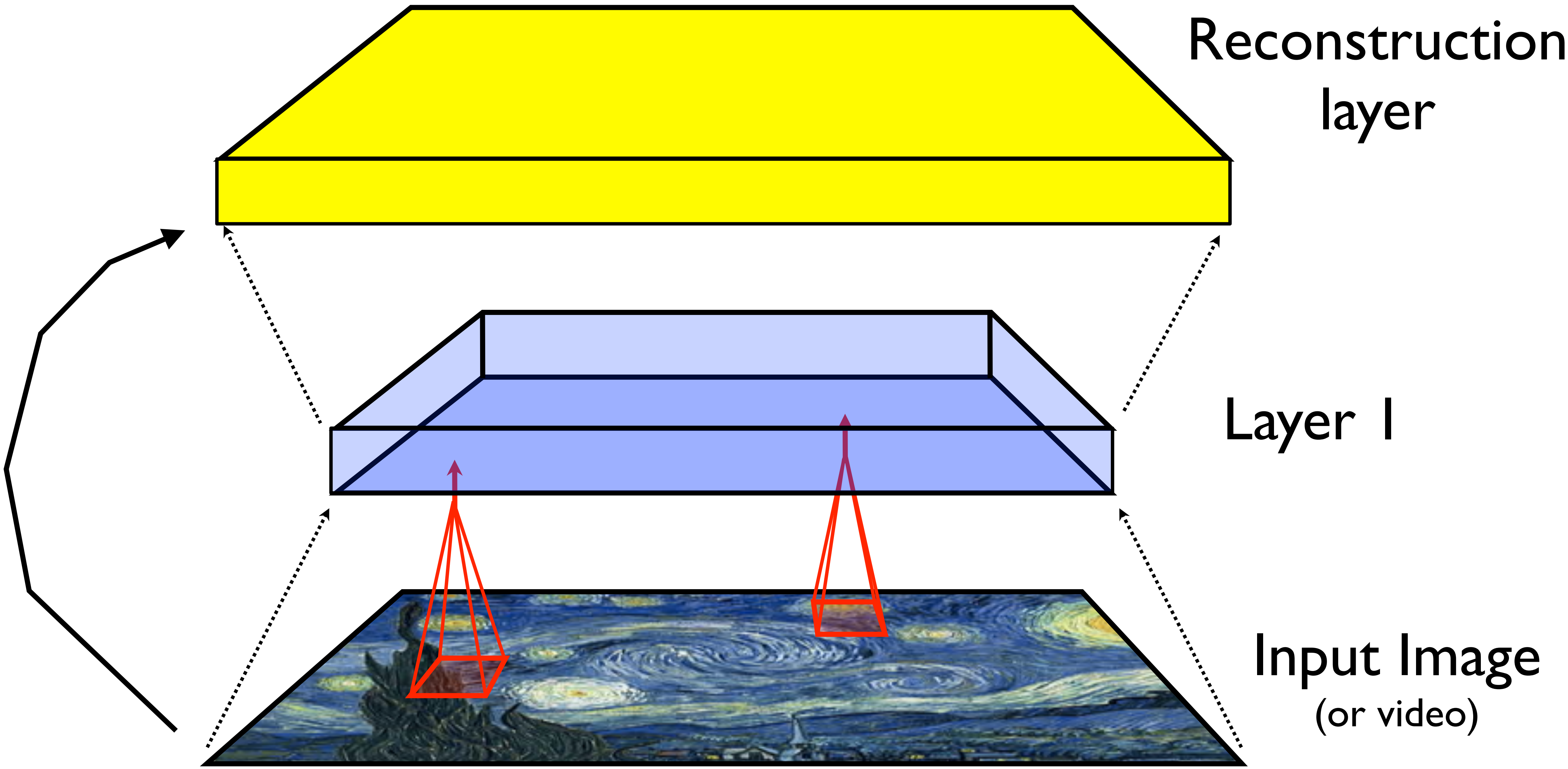


Deep Networks

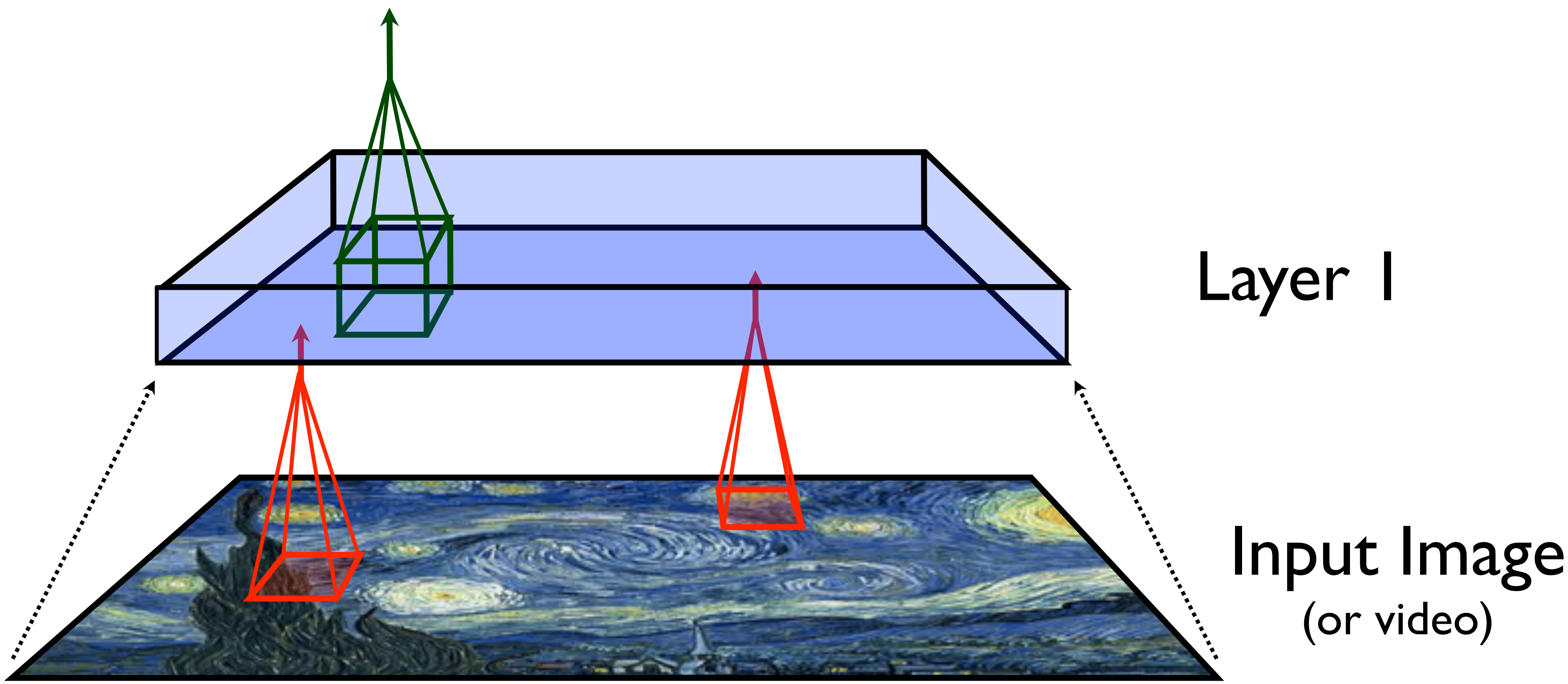


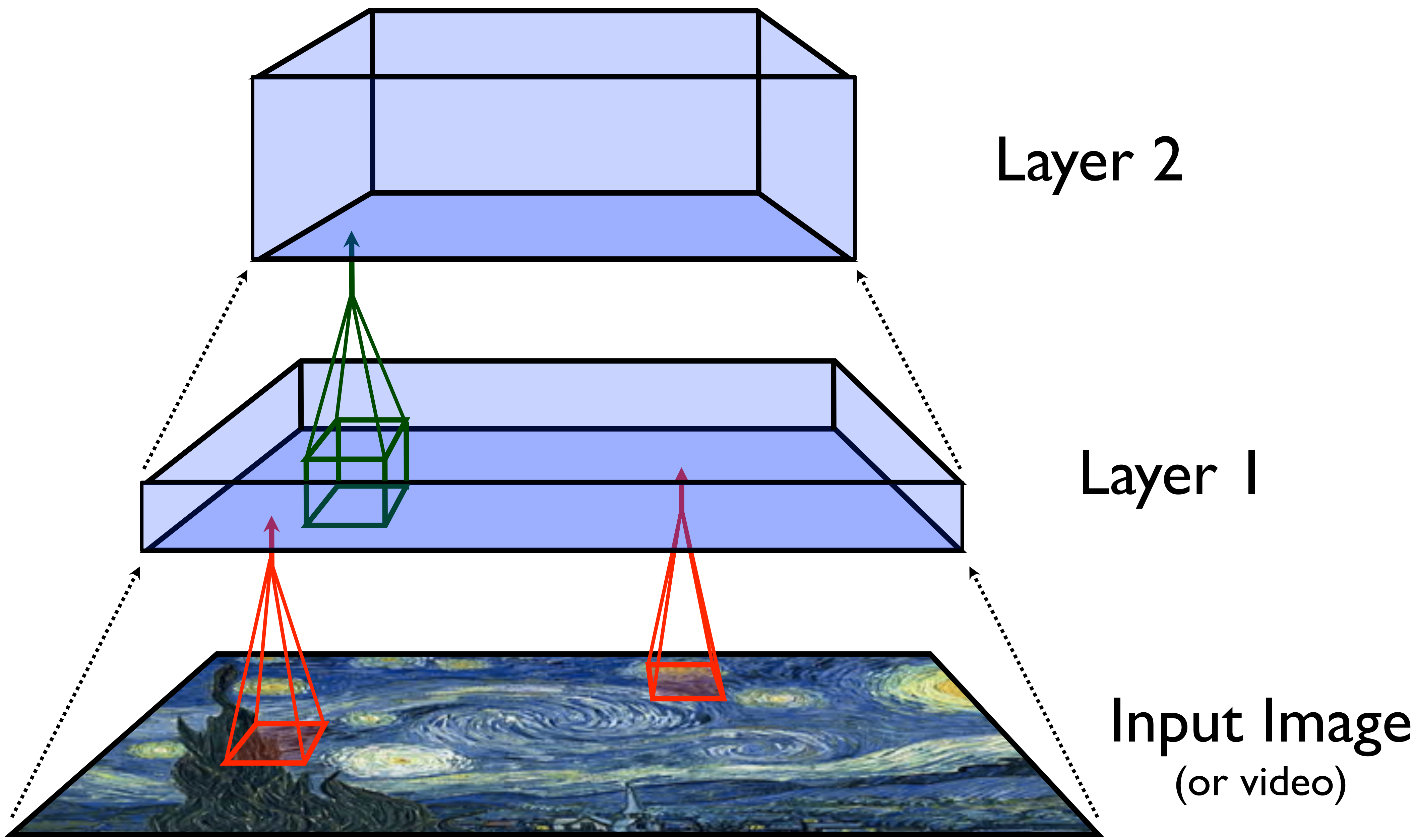
Unsupervised Training

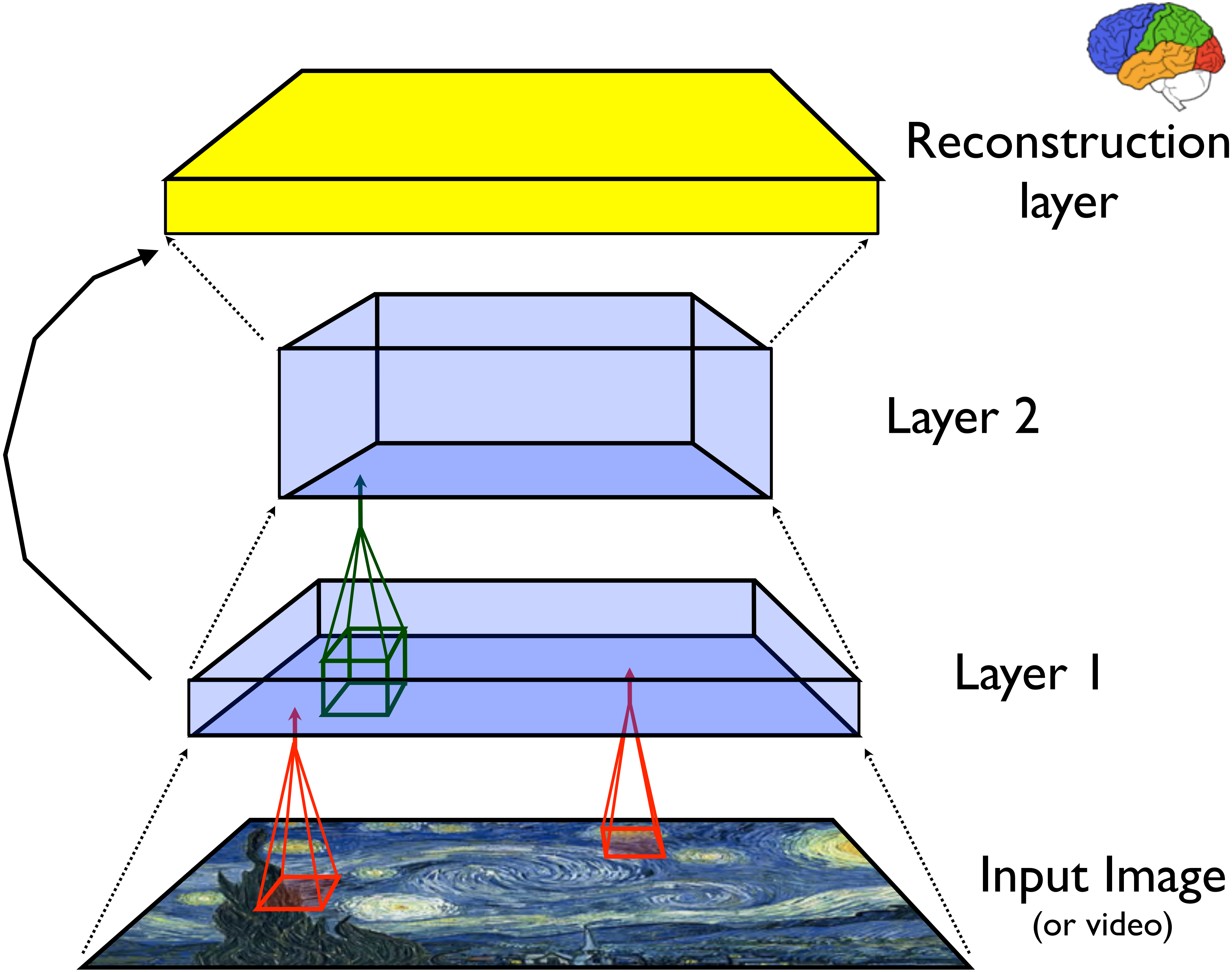
Core idea: try to reconstruct input from just the learned representation

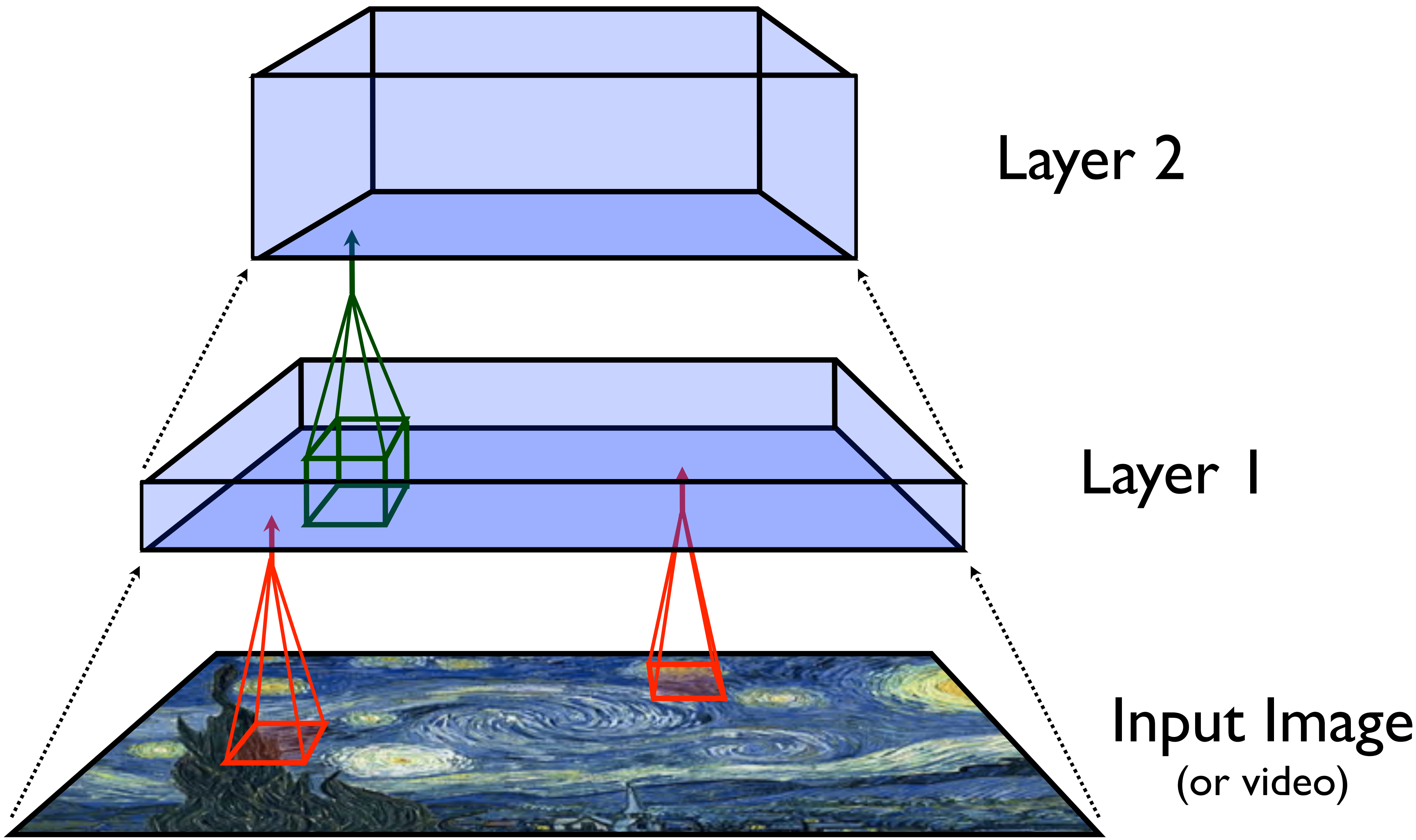


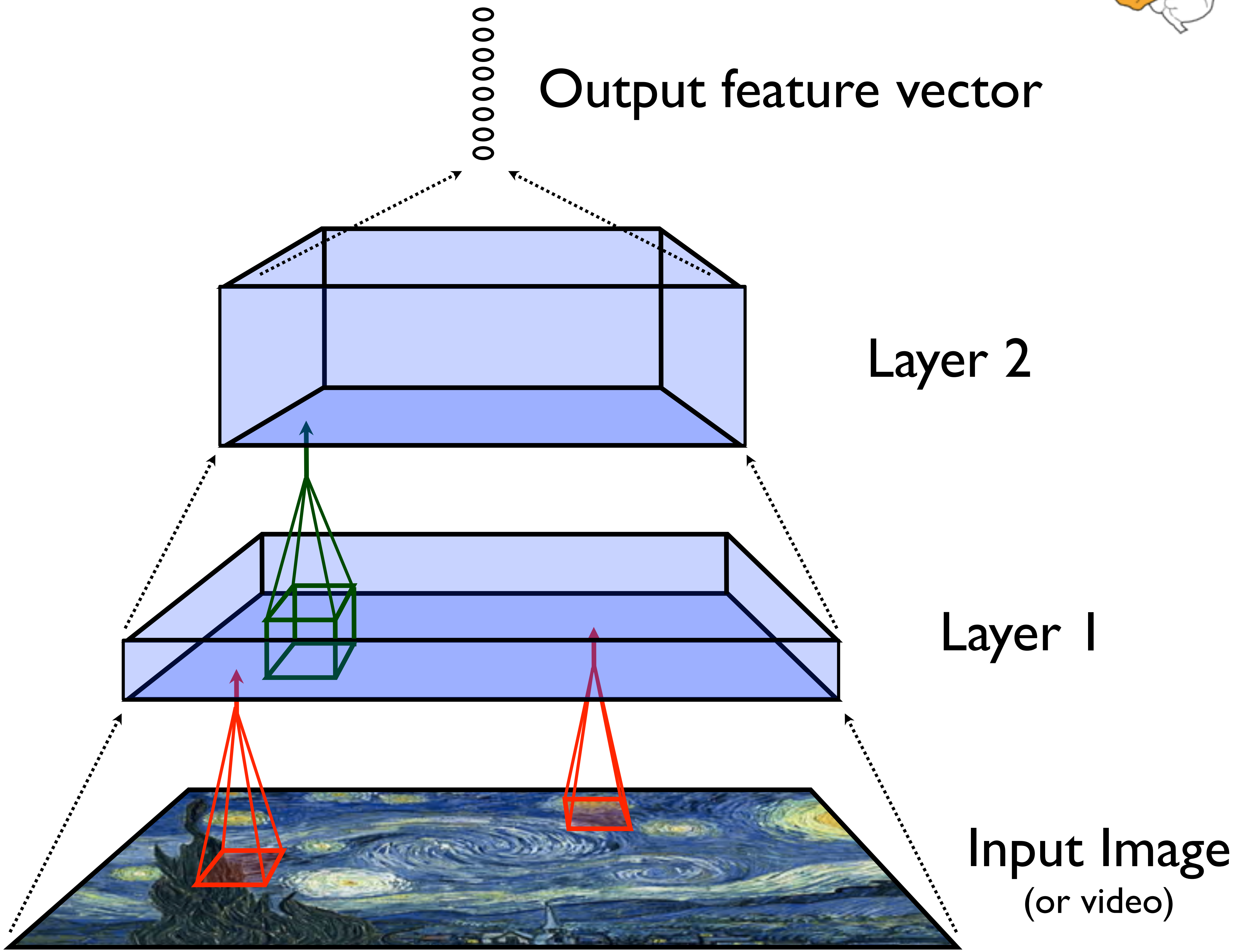
Due to Geoff Hinton, Yoshua Bengio, Andrew Ng, and others


Layer 1
Input Image
(or video)

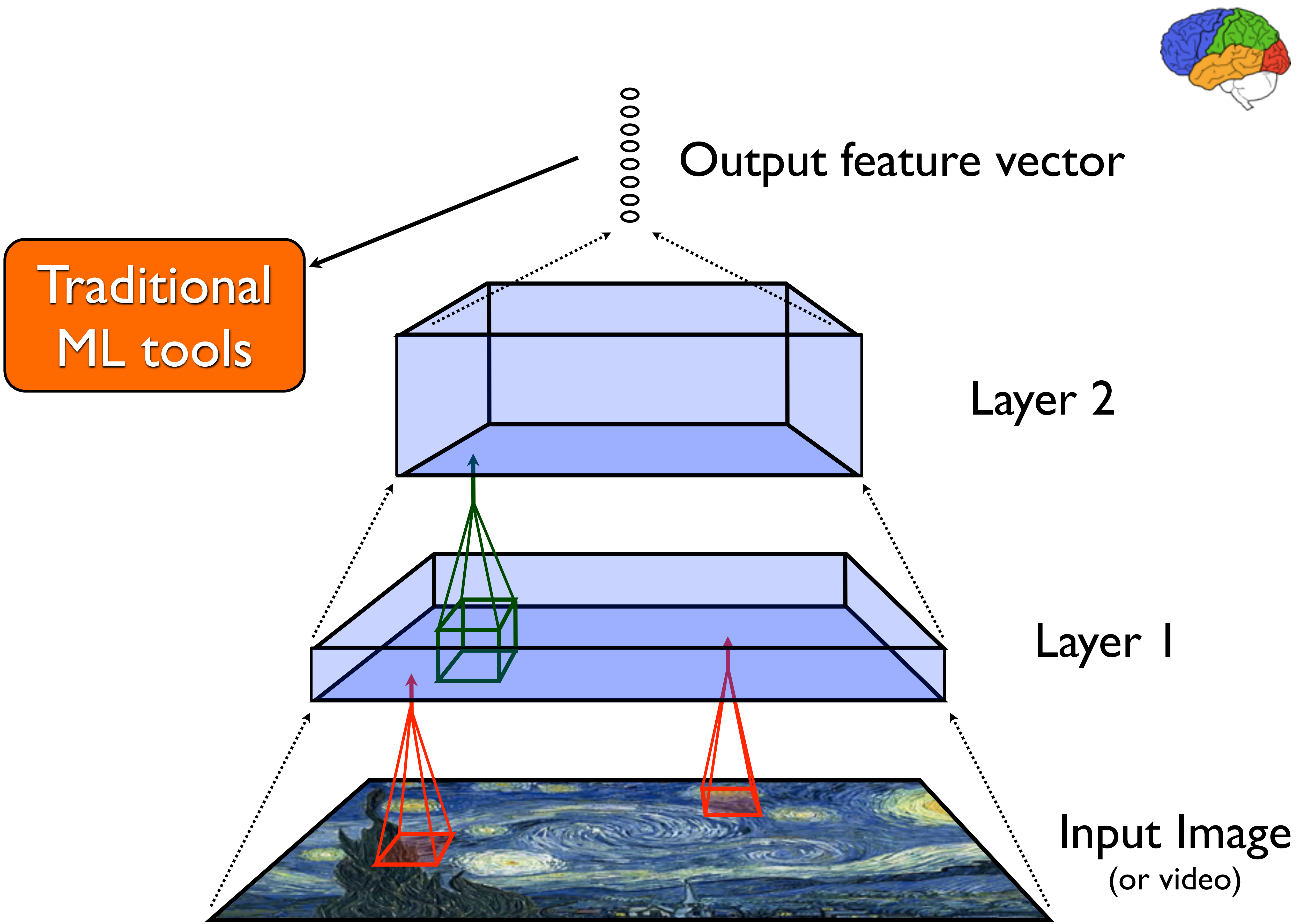

Layer 2
Layer 1
Input Image
(or video)


Reconstruction
layer
Layer 2
Layer 1
Input Image
(or video)


Layer 2
Layer 1
Input Image
(or video)

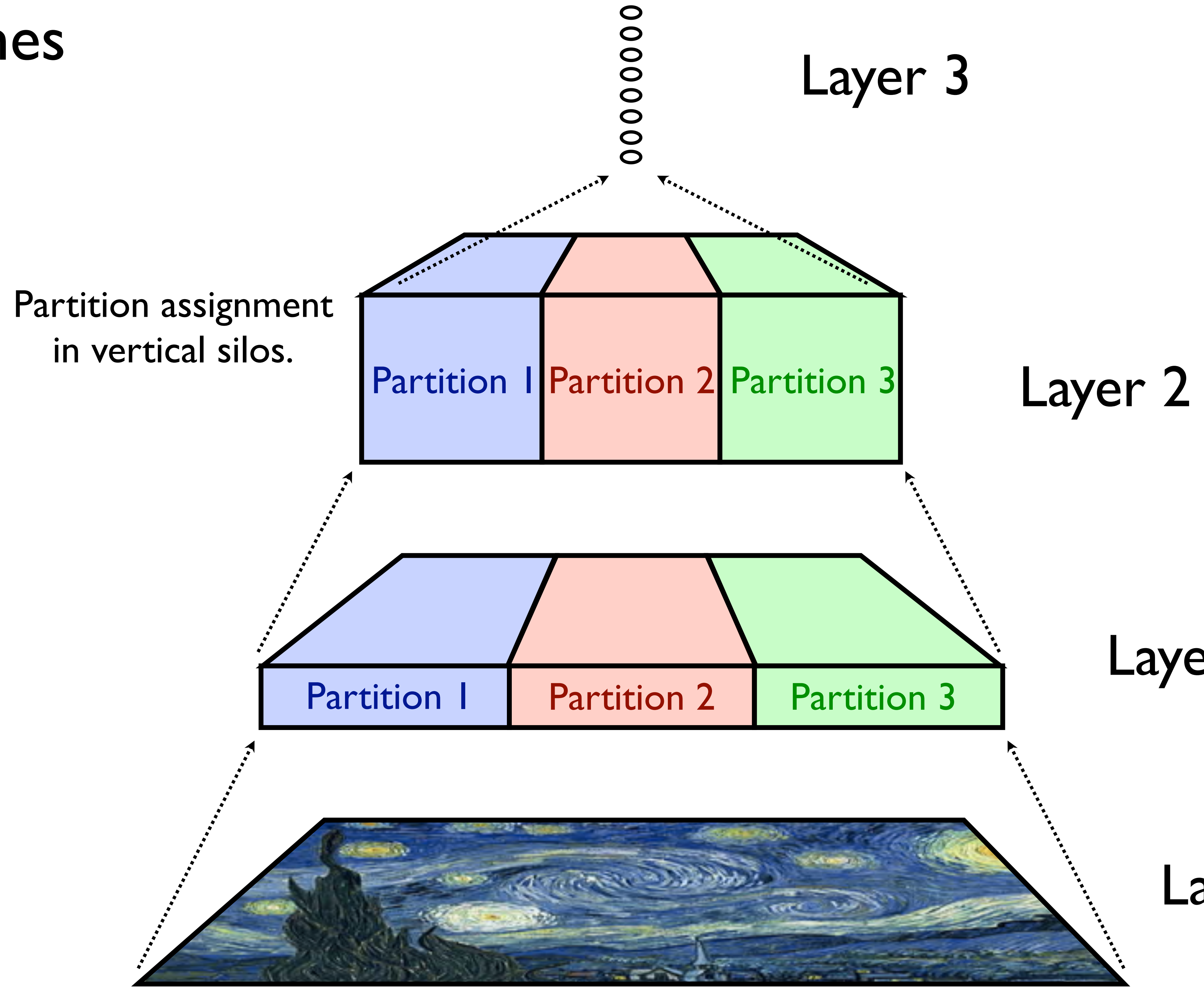


Output feature vector
Layer 2
Layer 1
Input Image
(or video)

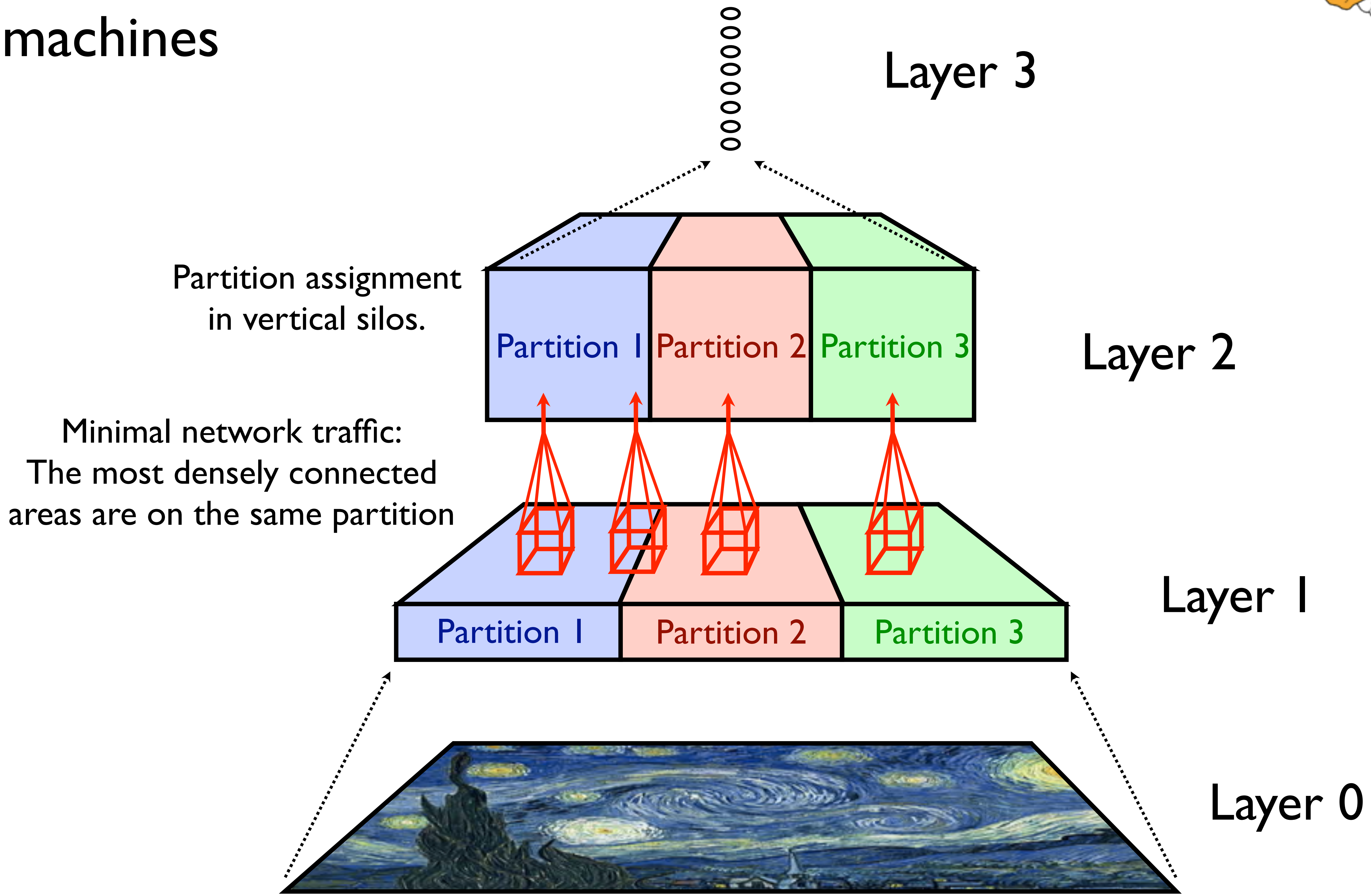


Output feature vector
Traditional ML tools
Layer 2
Layer 1
Input Image
(or video)

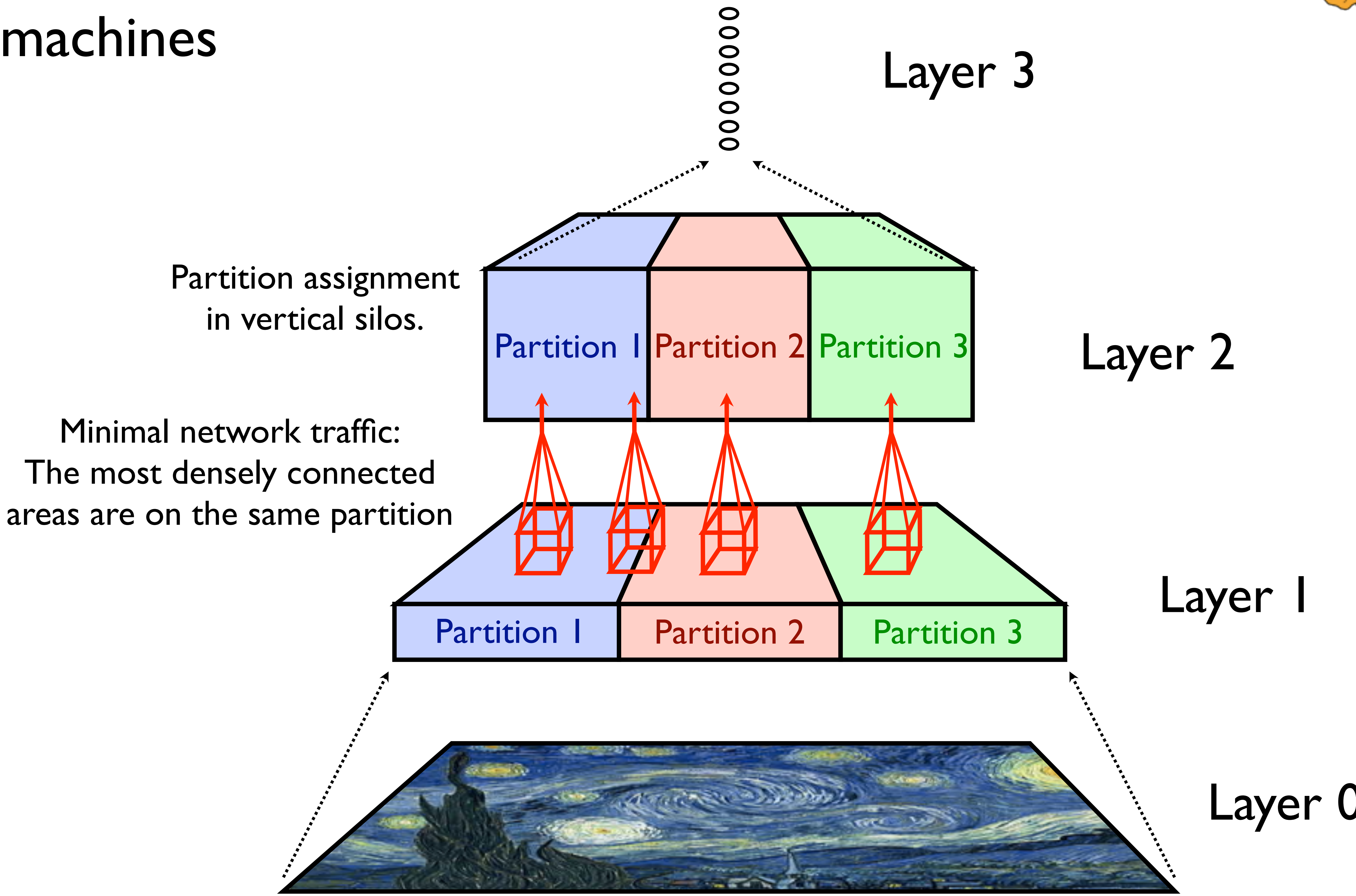
Partition model across machines



Partition model across machines

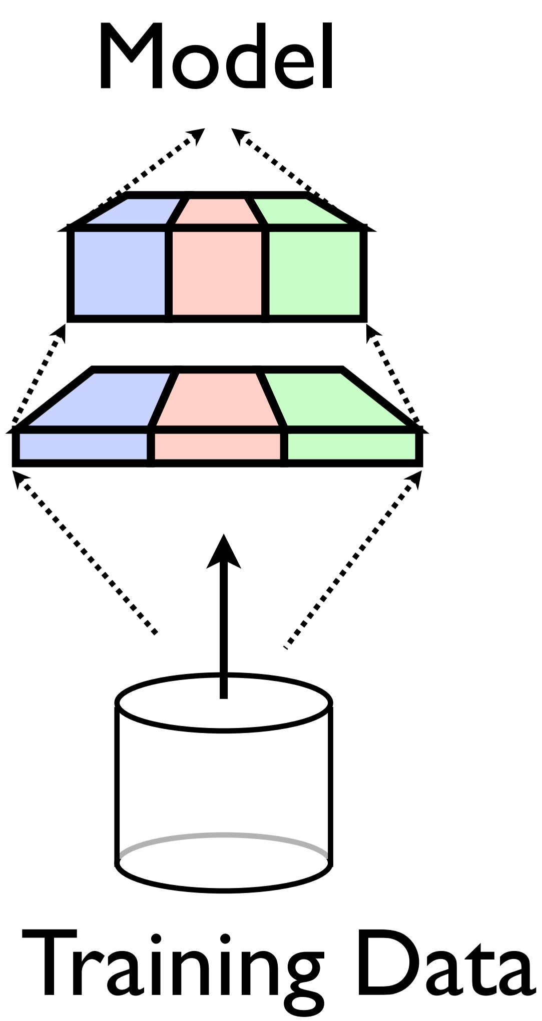


Partition model across machines



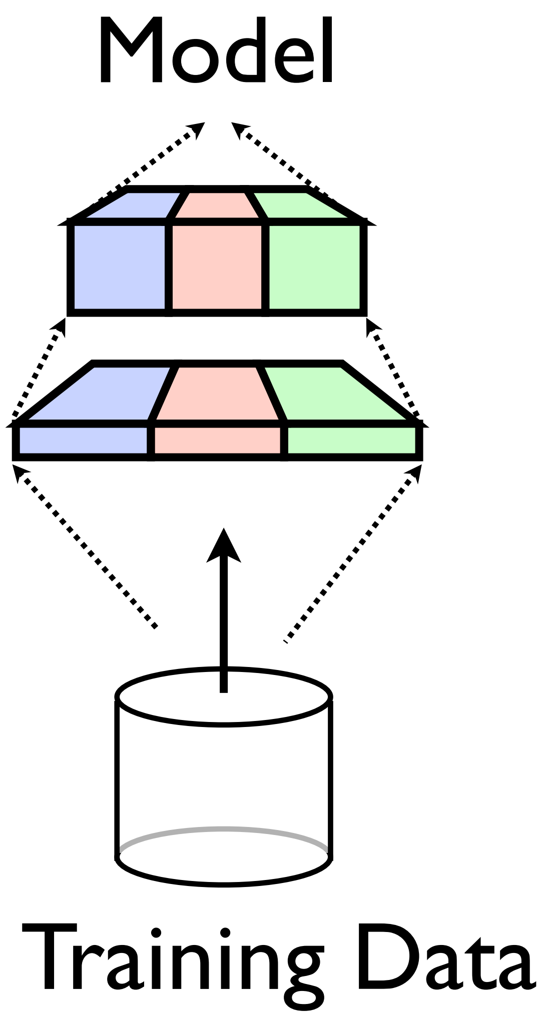
One replica of our biggest models: 144 machines, ~2300 cores

Basic Model Training



- Unsupervised or Supervised Objective
- Minibatch Stochastic Gradient Descent (SGD)
- Model parameters sharded by partition
- 10s, 100s, or 1000s of cores per model

Basic Model Training

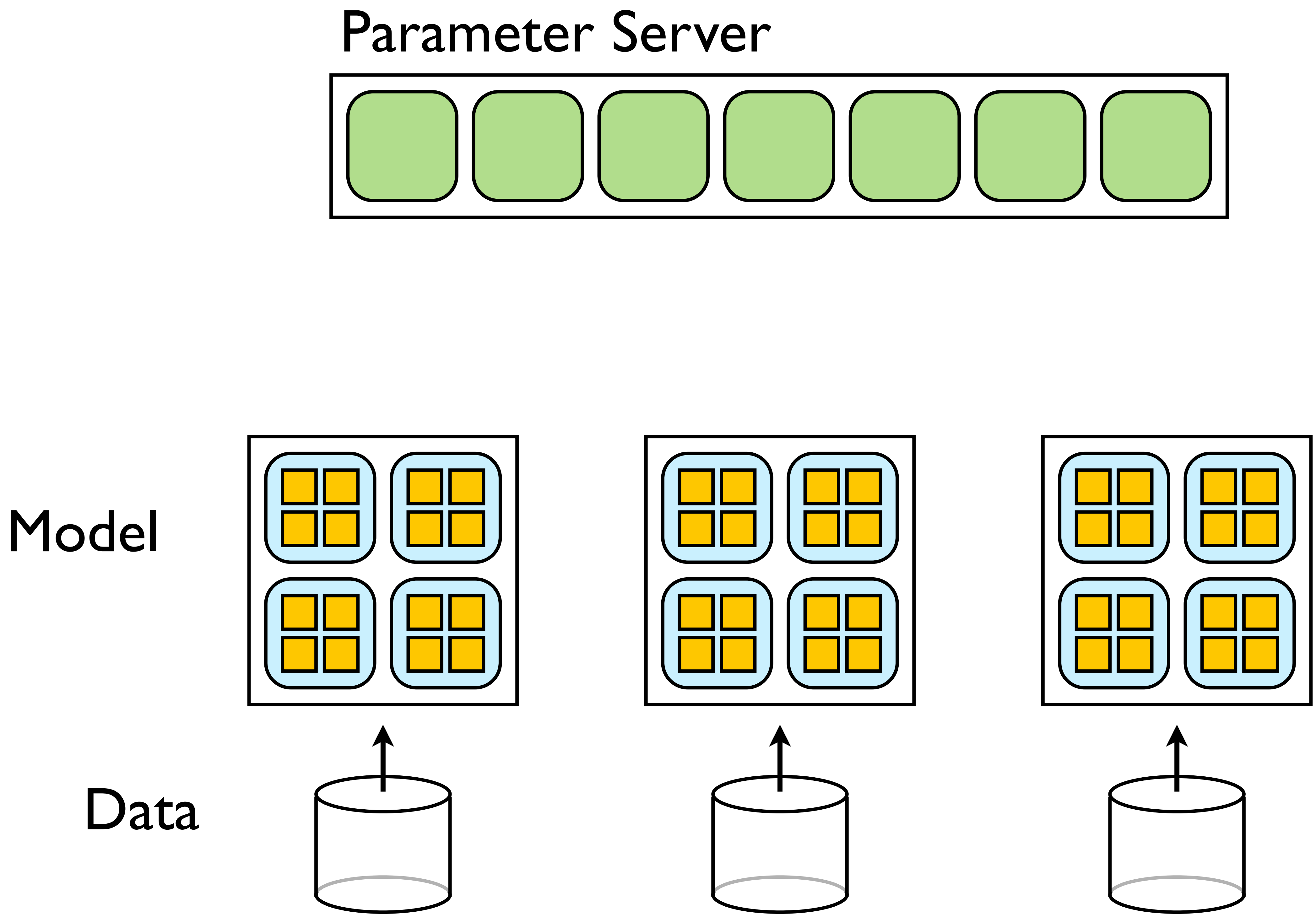


Making a single model bigger and faster is the right first step.

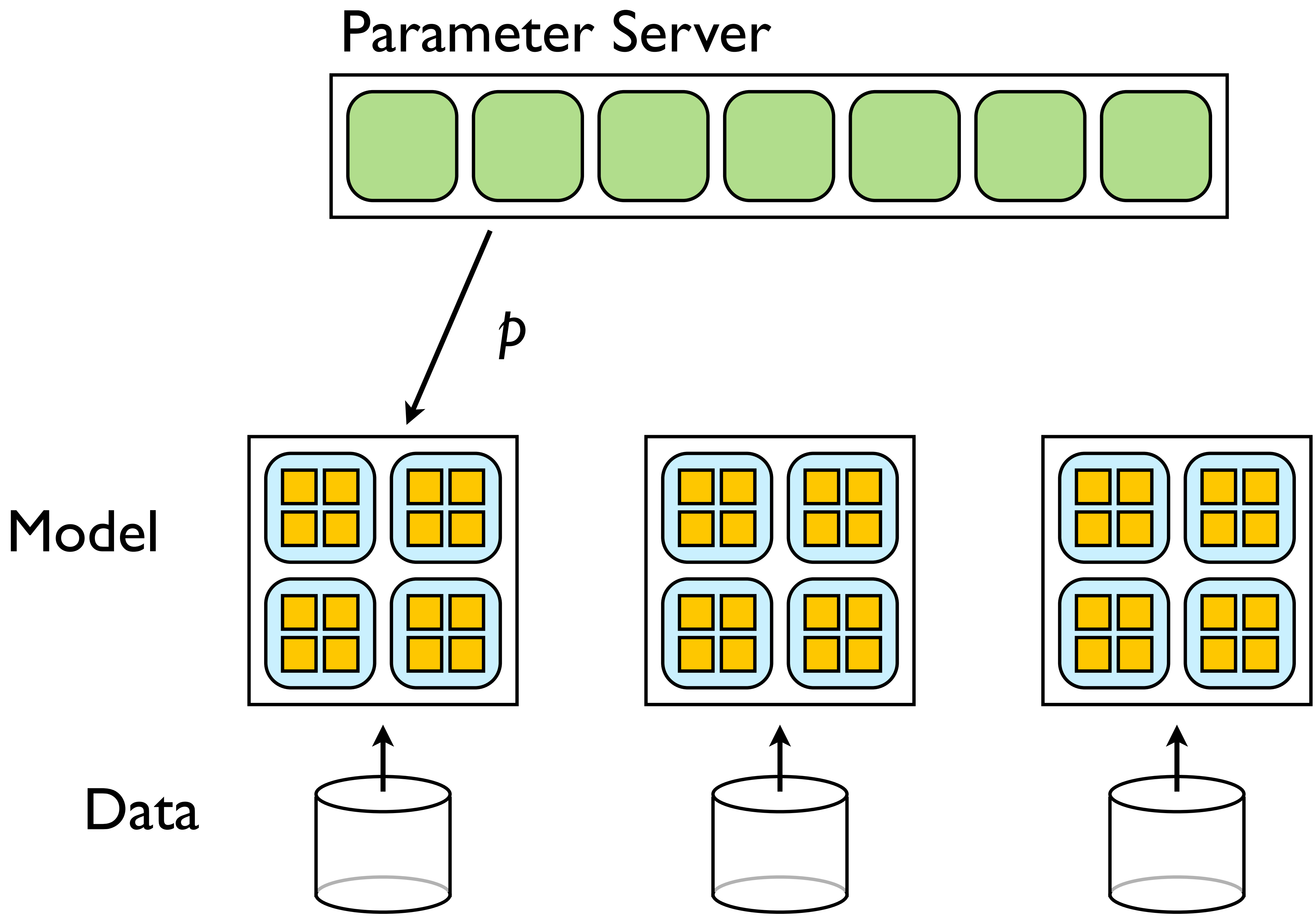
But training still slow with large data sets/model with a single model replica.

How can we add another dimension of parallelism, and have multiple model instances train on data in parallel?

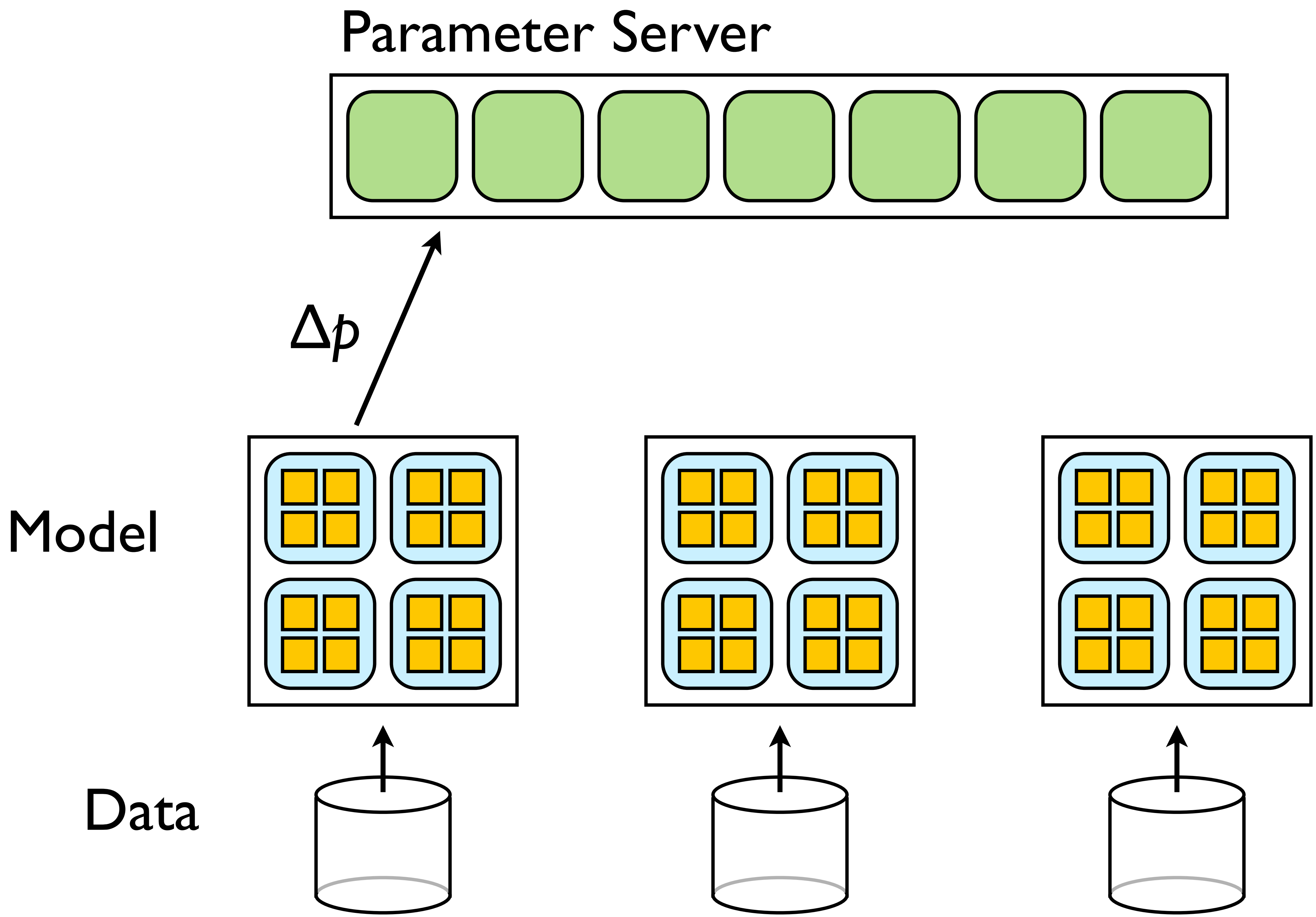
Asynchronous Distributed Stochastic Gradient Descent



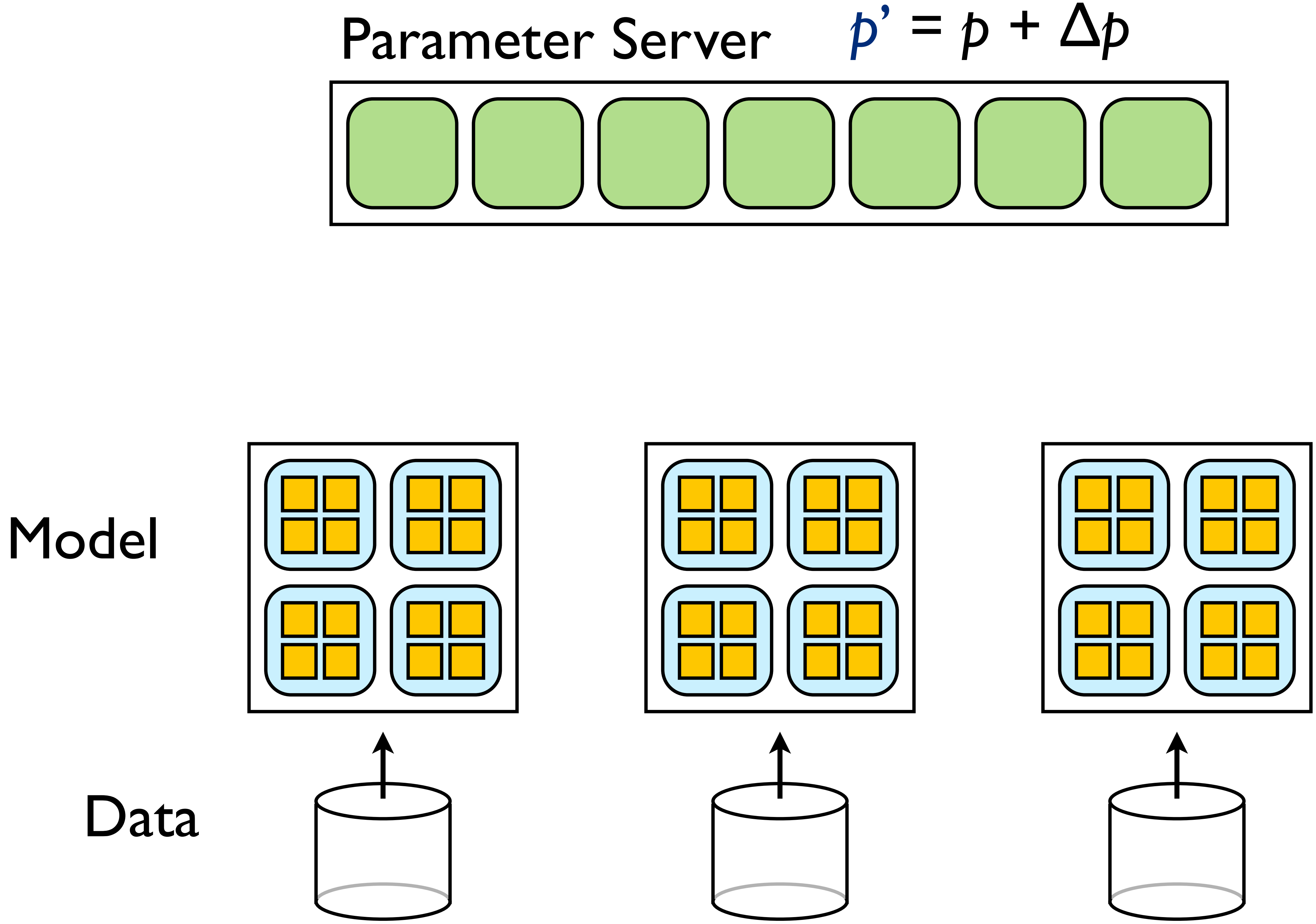
Asynchronous Distributed Stochastic Gradient Descent



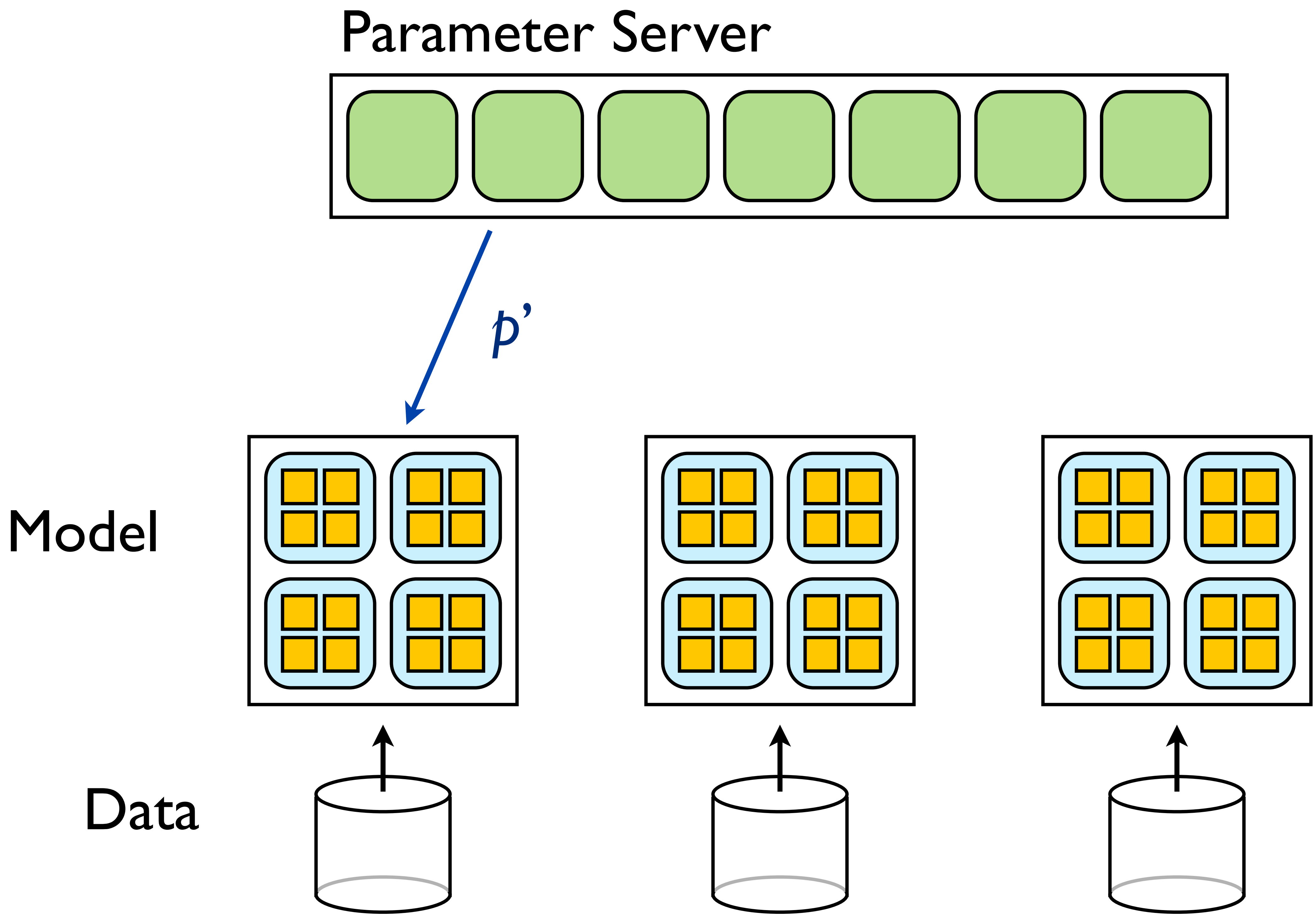
Asynchronous Distributed Stochastic Gradient Descent



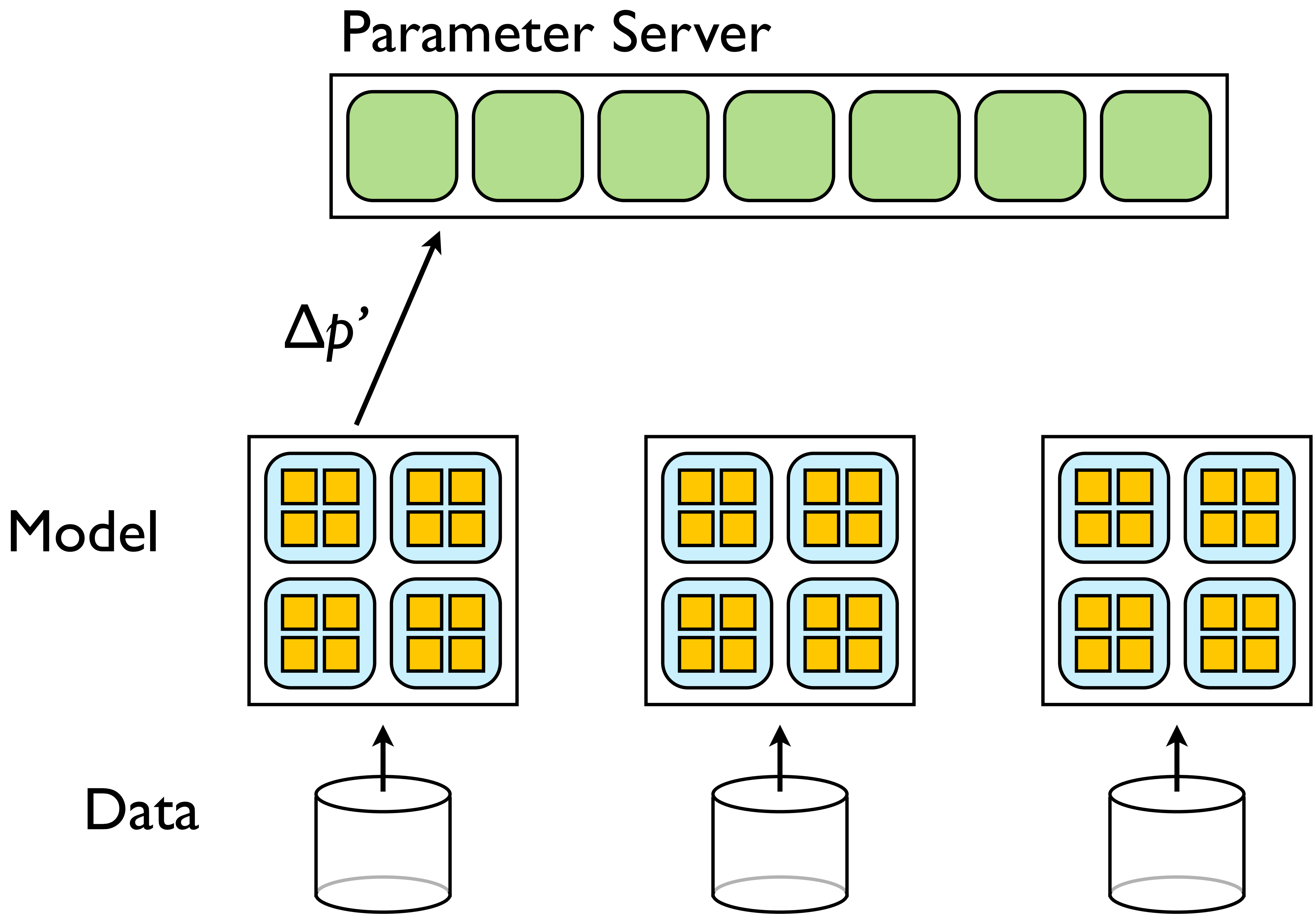
Asynchronous Distributed Stochastic Gradient Descent



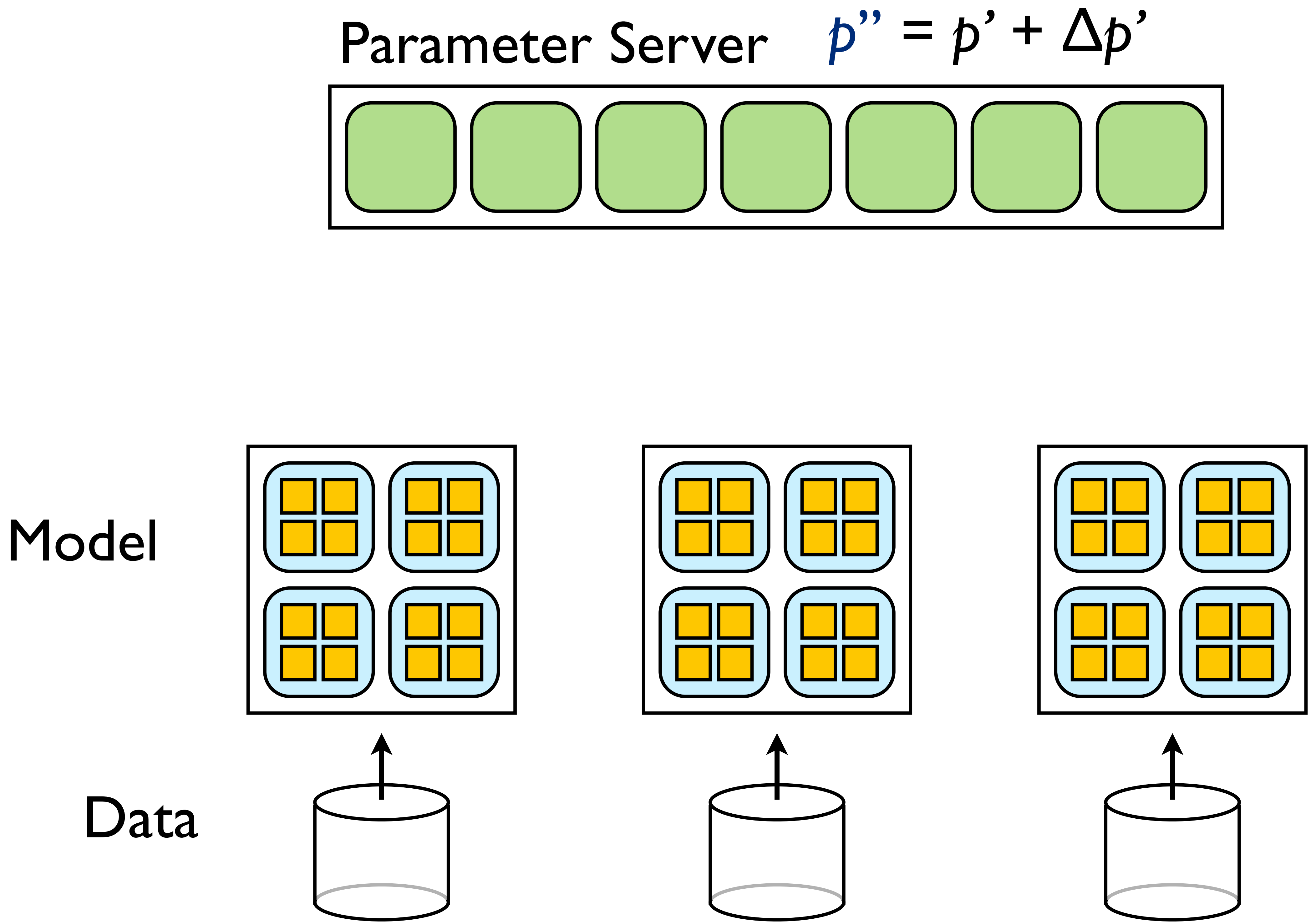
Asynchronous Distributed Stochastic Gradient Descent



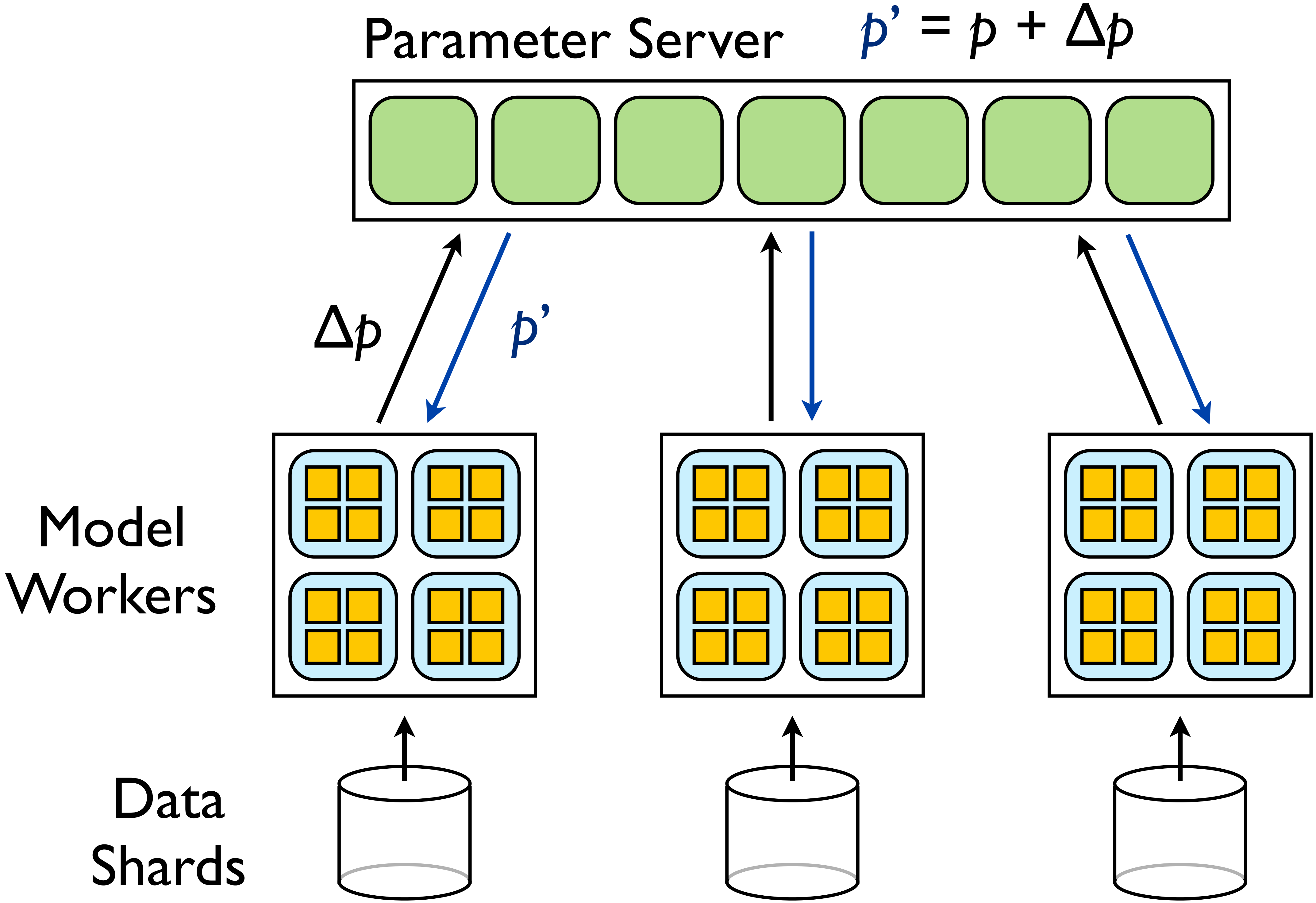
Asynchronous Distributed Stochastic Gradient Descent



Asynchronous Distributed Stochastic Gradient Descent



Asynchronous Distributed Stochastic Gradient Descent



Training System

- Some aspects of asynchrony and distribution similar to some recent work:

  Slow Learners are Fast
  John Langford, Alexander J. Smola, Martin Zinkevich, NIPS 2009

  Distributed Delayed Stochastic Optimization
  Alekh Agarwal, John Duchi, NIPS 2011

  Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent
  Feng Niu, Benjamin Recht, Christopher Re, Stephen J. Wright, NIPS 2011

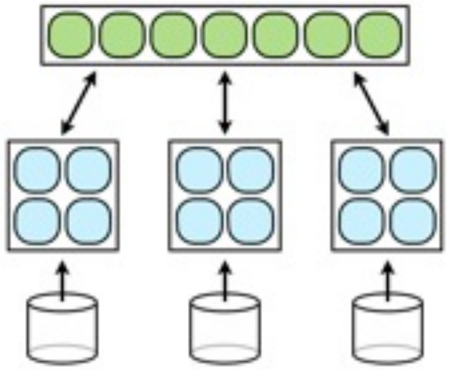
- Details of our system to appear:

  [*Large Scale Distributed Deep Networks*, Dean et al., to appear in NIPS 2012]

Deep Learning Systems Tradeoffs

- Lots of tradeoffs can be made to improve performance. Which ones are possible without hurting learning performance too much?
- For example:
  - Use lower precision arithmetic
  - Send 1 or 2 bits instead of 32 bits across network
  - Drop results from slow partitions
- What's the right hardware for training and deploying these sorts of systems?
  - GPUs? FPGAs? Lossy computational devices?

Applications



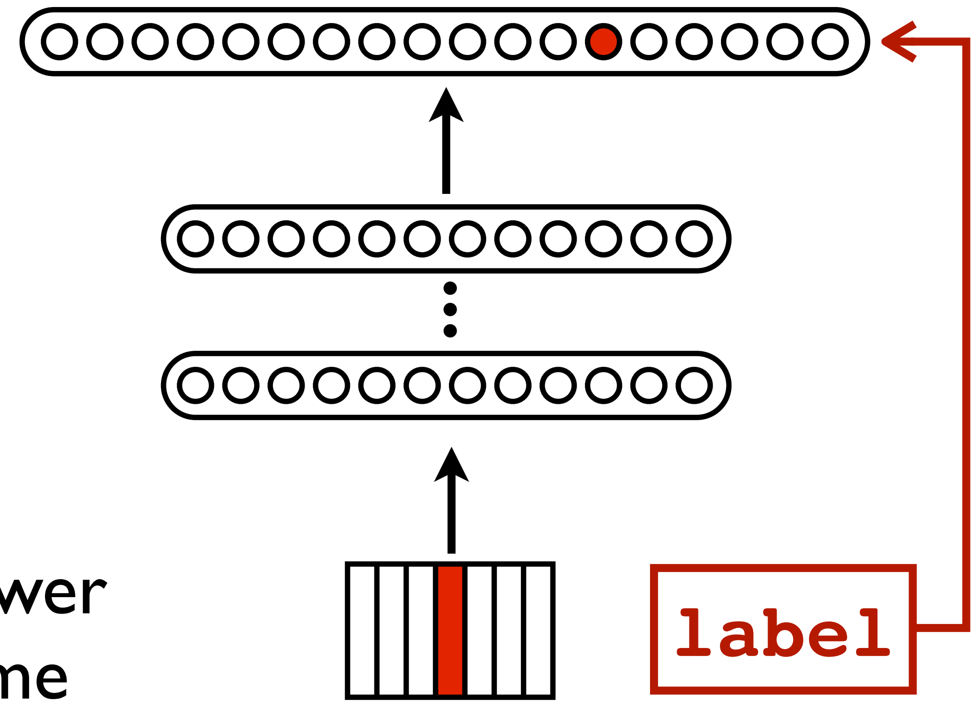
- Acoustic Models for Speech
- Unsupervised Feature Learning for Still Images
- Neural Language Models

Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers
of a few thousand nodes each.

11 Frames of 40-value Log Energy Power
Spectra and the label for central frame



Close collaboration with Google Speech team

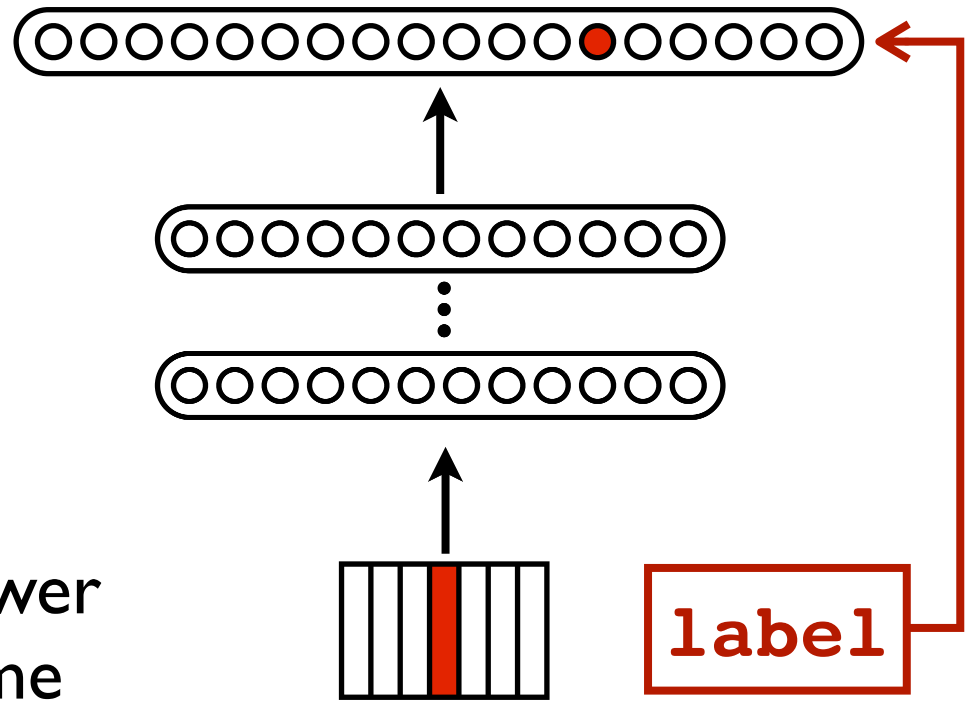
Trained in <5 days on cluster of 800 machines

Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers
of a few thousand nodes each.

11 Frames of 40-value Log Energy Power
Spectra and the label for central frame



Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines

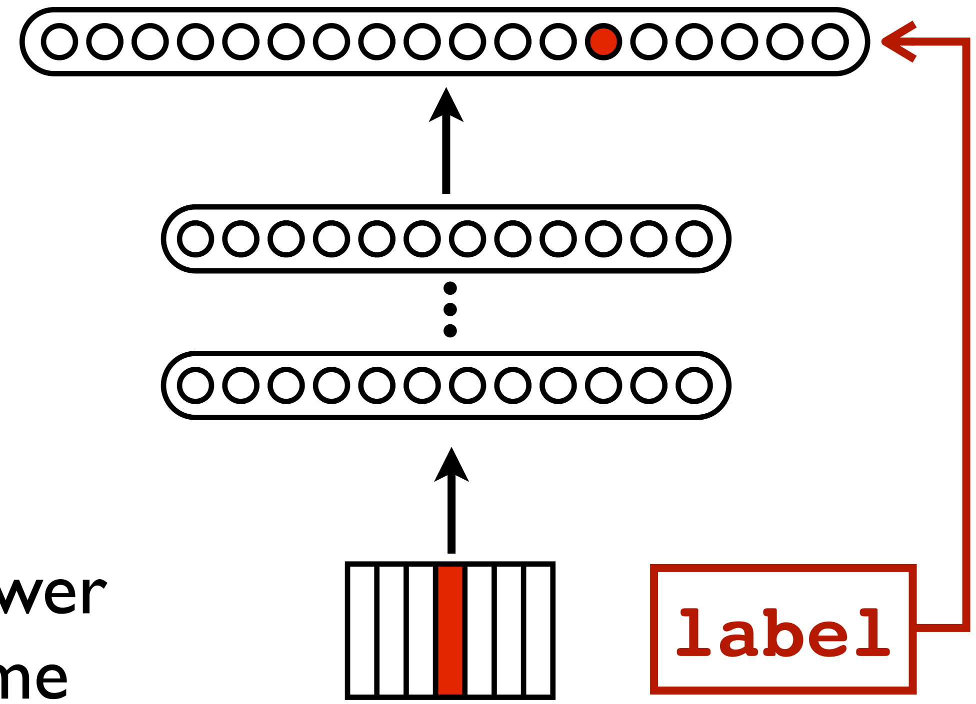
Major reduction in Word Error Rate
("equivalent to 20 years of speech research")

Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers
of a few thousand nodes each.

11 Frames of 40-value Log Energy Power
Spectra and the label for central frame



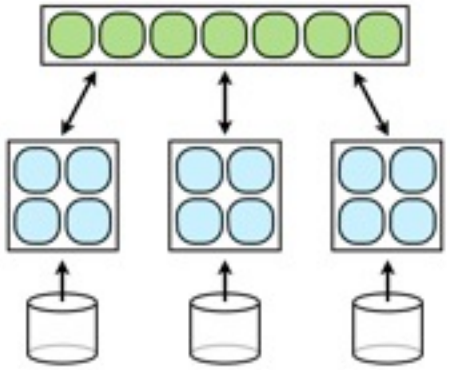
Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines

Major reduction in Word Error Rate
("equivalent to 20 years of speech research")

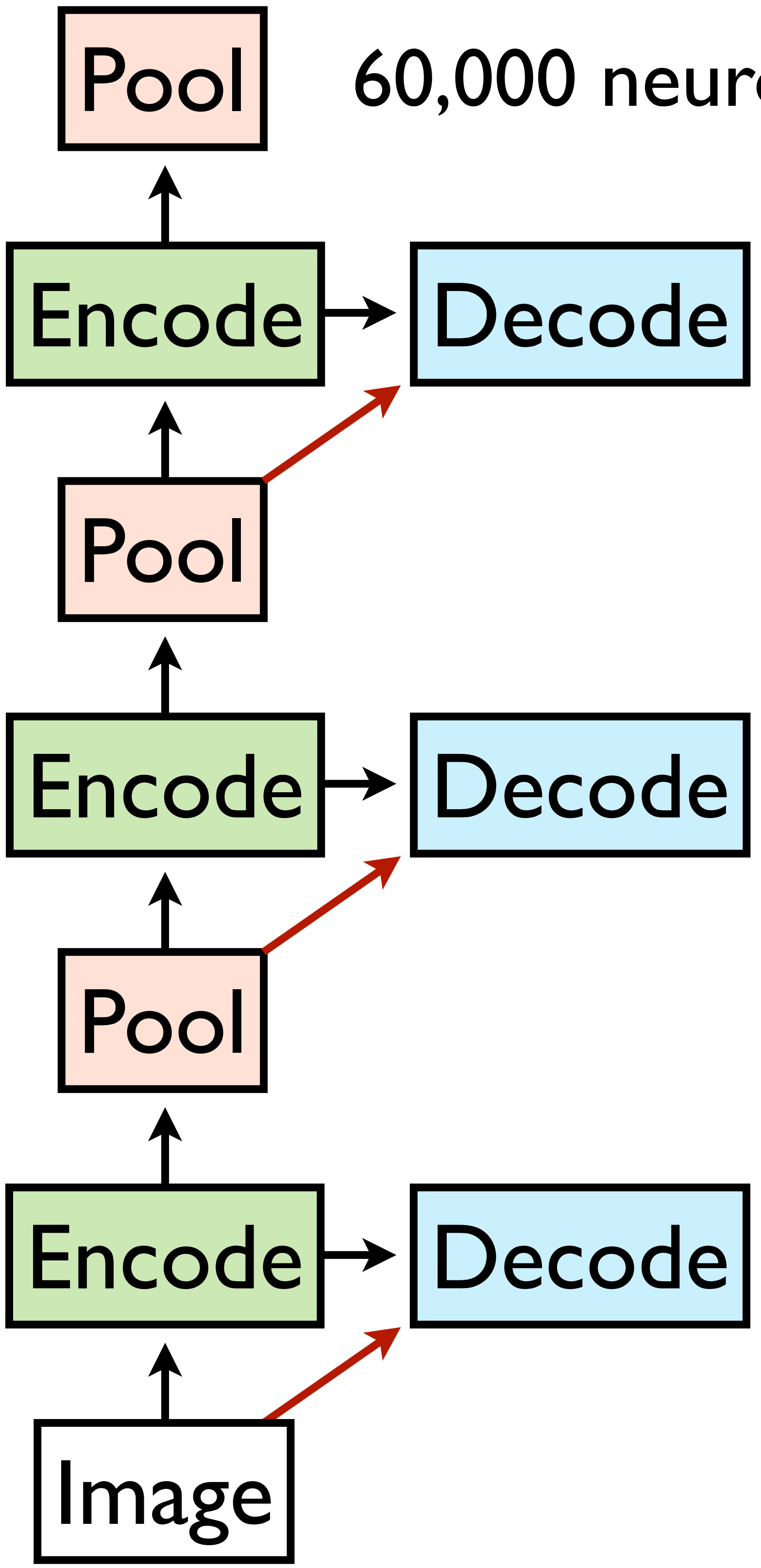
Deployed in Jellybean release of Android

Applications



- Acoustic Models for Speech
- Unsupervised Feature Learning for Still Images
- Neural Language Models

Purely Unsupervised Feature Learning in Images



60,000 neurons at top level

- 1.15 billion parameters (50x larger than largest deep network in the literature)

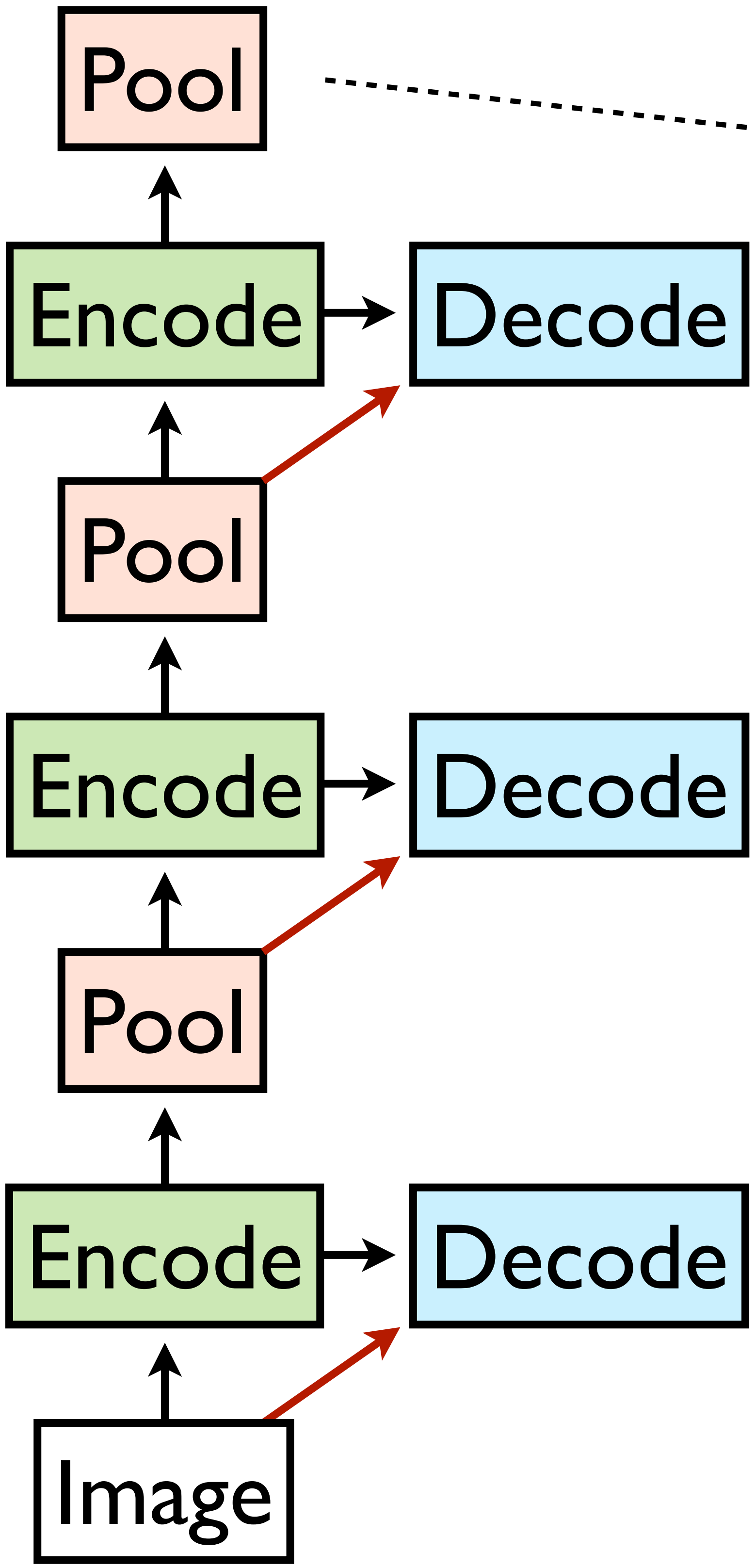
- Trained on 16k cores for 1 week using Async-SGD

- Do **unsupervised** training on one frame from each of 10 million YouTube videos (200x200 pixels)

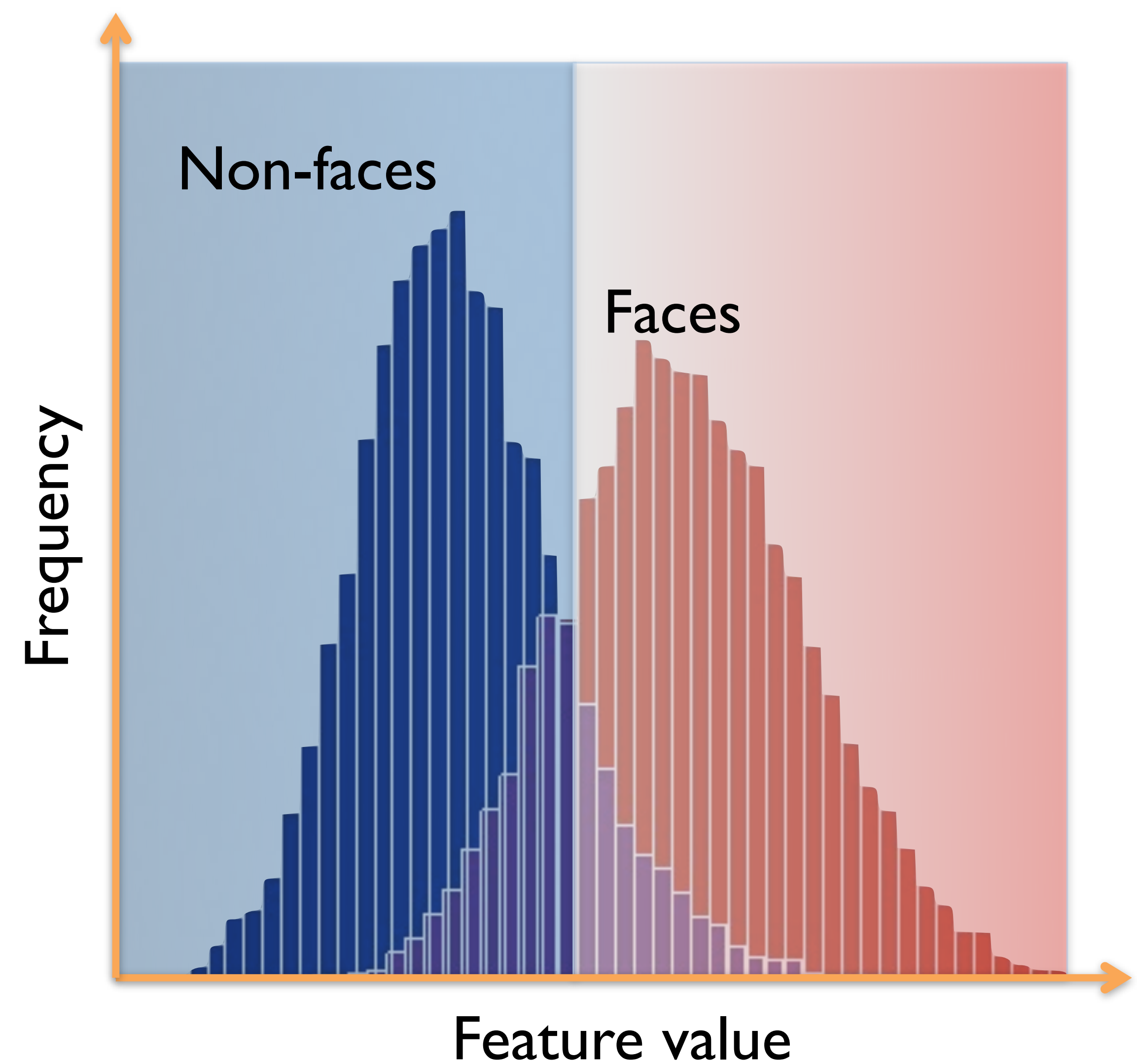
- No labels!

Details in our ICML paper [Le et al. 2012]

Purely Unsupervised Feature Learning in Images



Top level neurons seem to discover high-level concepts. For example, one neuron is a decent face detector:



Purely Unsupervised Feature Learning in Images

Most face-selective neuron

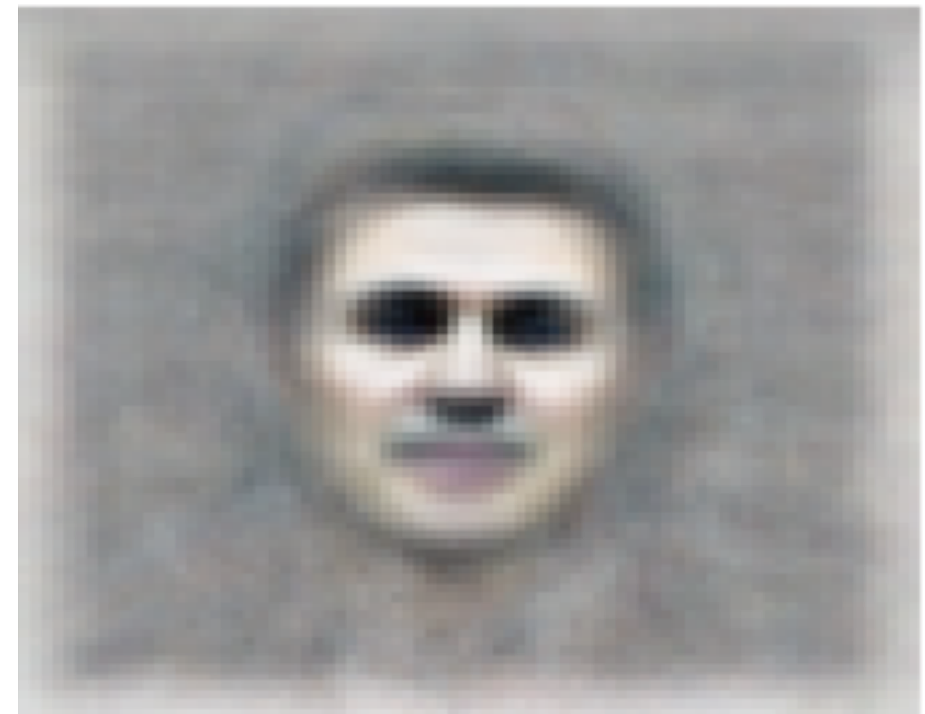
Top 48 stimuli from the test set

Purely Unsupervised Feature Learning in Images

Most face-selective neuron

Top 48 stimuli from the test set

Optimal stimulus
by numerical optimization



Purely Unsupervised Feature Learning in Images

It is YouTube... We also have a cat neuron!

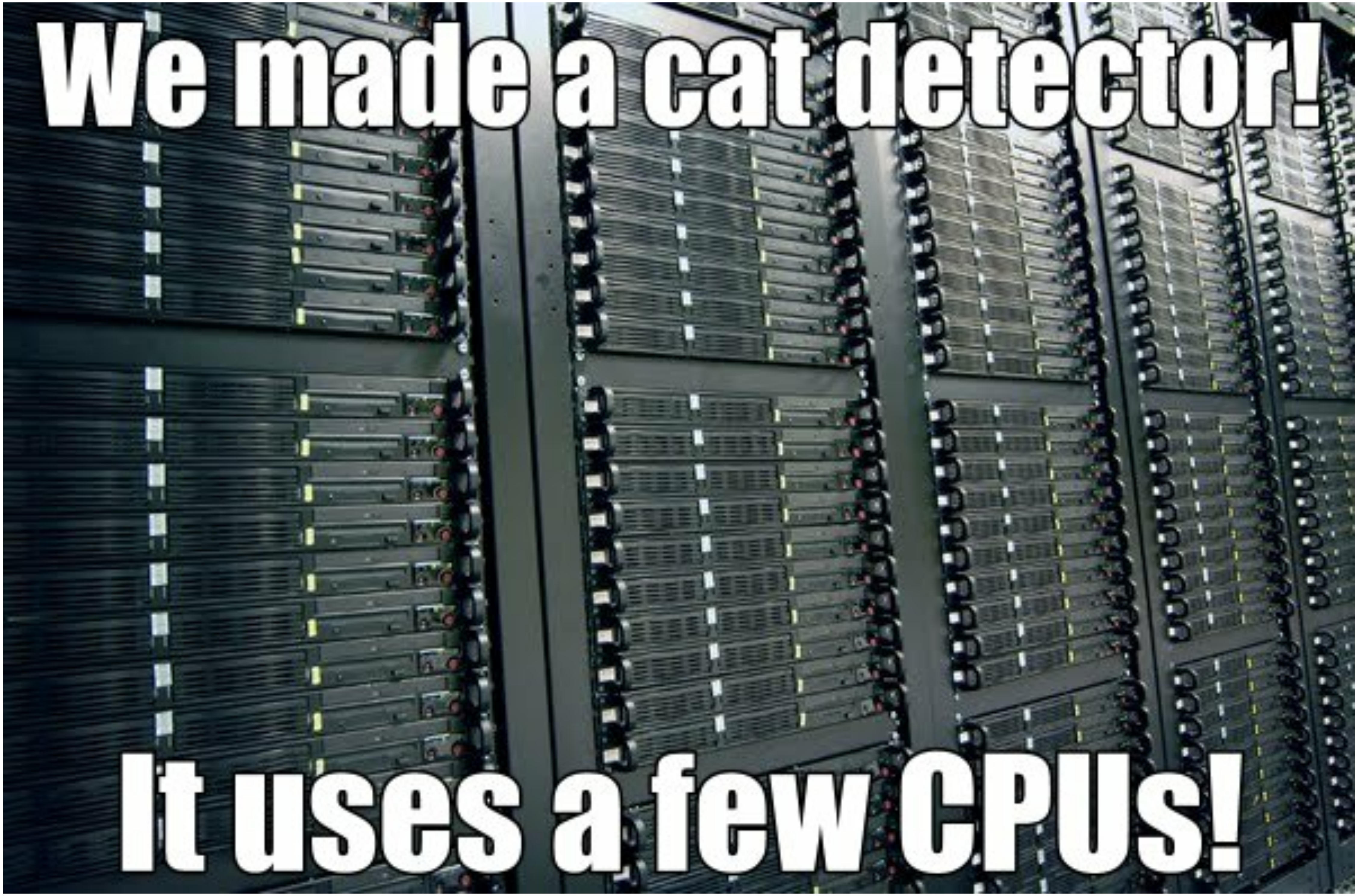
Top stimuli from the test set

Purely Unsupervised Feature Learning in Images

It is YouTube... We also have a cat neuron!

Top stimuli from the test set

Optimal stimulus


We made a cat detector!
It uses a few CPUs!

Semi-supervised Feature Learning in Images

Are the higher-level representations learned by unsupervised training a useful starting point for supervised training?

We do have *some* labeled data, so let's fine tune this same network for a challenging image classification task.

Semi-supervised Feature Learning in Images

Are the higher-level representations learned by unsupervised training a useful starting point for supervised training?

We do have *some* labeled data, so let's fine tune this same network for a challenging image classification task.

ImageNet:

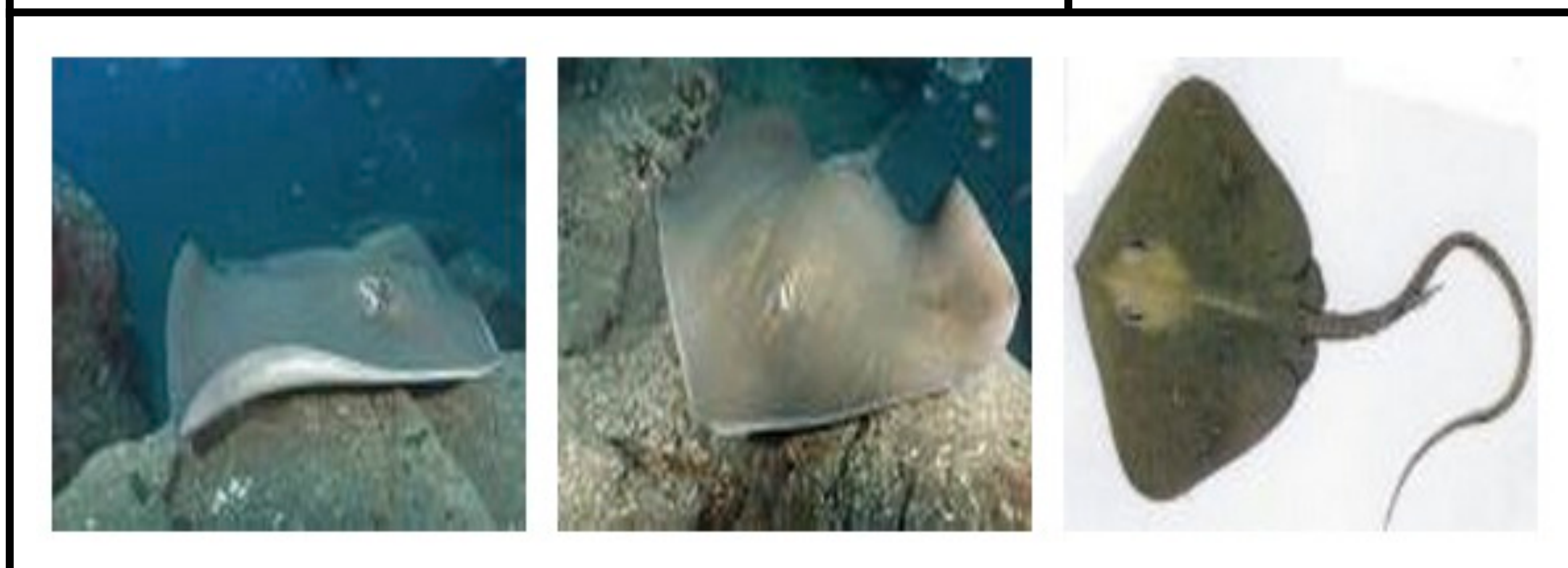
- 16 million images
- ~21,000 categories
- Recurring academic competitions

Aside: 20,000 is a lot of categories....

```
01496331 electric ray, crampfish, numbfish, torpedo
01497118 sawfish
01497413 smalltooth sawfish, Pristis pectinatus
01497738 guitarfish
01498041 stingray
01498406 roughtail stingray, Dasyatis centroura
01498699 butterfly ray
01498989 eagle ray
01499396 spotted eagle ray, spotted ray, Aetobatus narinari
01499732 cownose ray, cow-nosed ray, Rhinoptera bonasus
01500091 manta, manta ray, devilfish
01500476 Atlantic manta, Manta birostris
01500854 devil ray, Mobula hypostoma
01501641 grey skate, gray skate, Raja batis
01501777 little skate, Raja erinacea
01501948 thorny skate, Raja radiata
01502101 barndoor skate, Raja laevis
01503976 dickeybird, dickey-bird, dickybird, dicky-bird
01504179 fledgling, fledgeling
01504344 nestling, baby bird
```

Aside: 20,000 is a lot of categories....

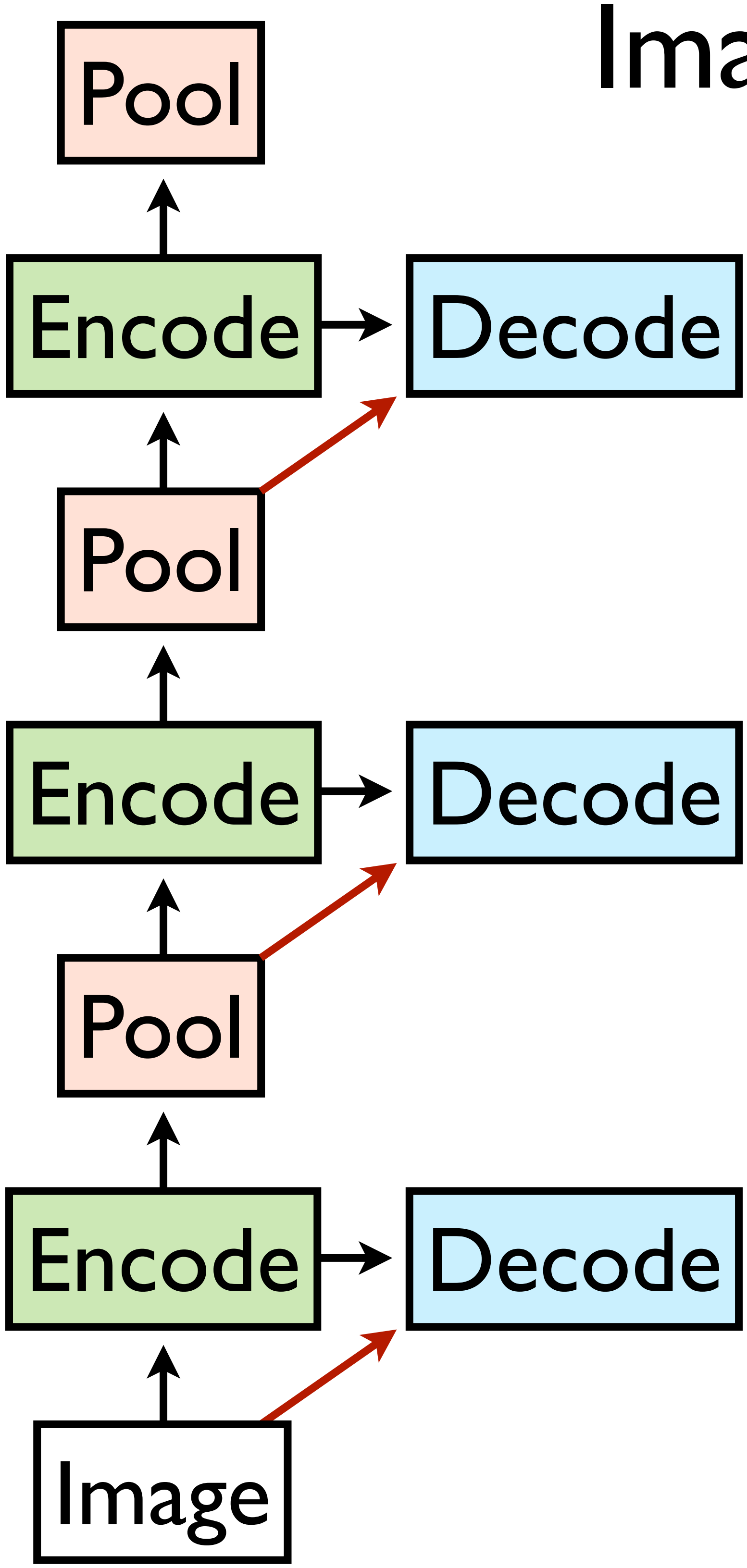
roughtail stingray



0149 , numbfish, torpedo
0149
0149 us
0149
0149
0149 ra
0149
0149
0149 atus narinari
01499732 cownose ray, cow-nosed ray, Rhinoptera bonasus
01500091 manta, ma fish
01500476 Atlantic
01500854 devil ray
01501641 grey skat
01501777 little sk
01501948 thorny sk
01502101 barndoor
01503976 dickeybir
01504179 fledgling
01504344 nestling, baby bird

manta ray

Semi-supervised Feature Learning in Images



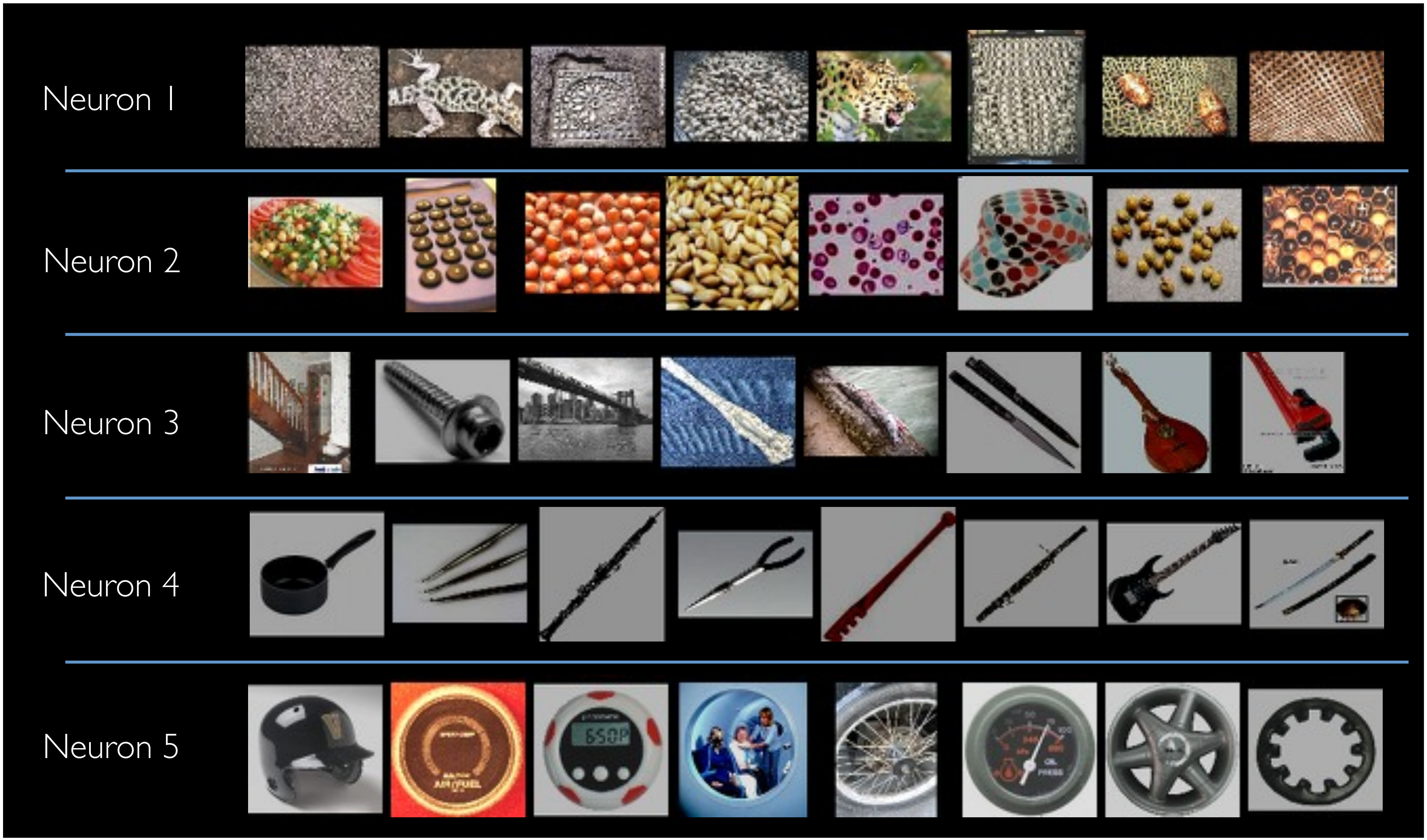
ImageNet Classification Results:

ImageNet 2011 (20k categories)

- Chance: 0.005%
- Best reported: 9.5%
- Our network: 16% (+70% relative)

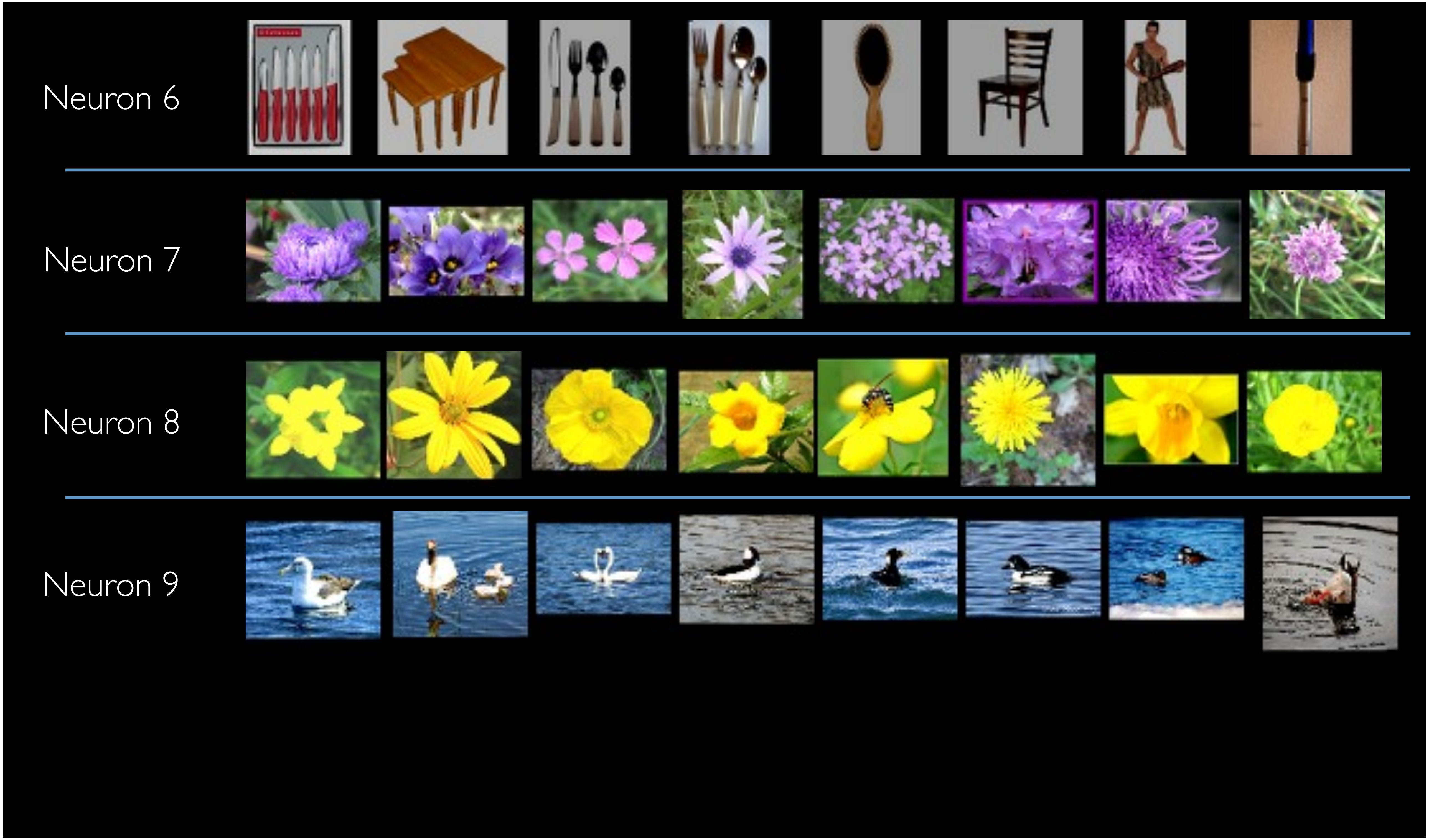
Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:



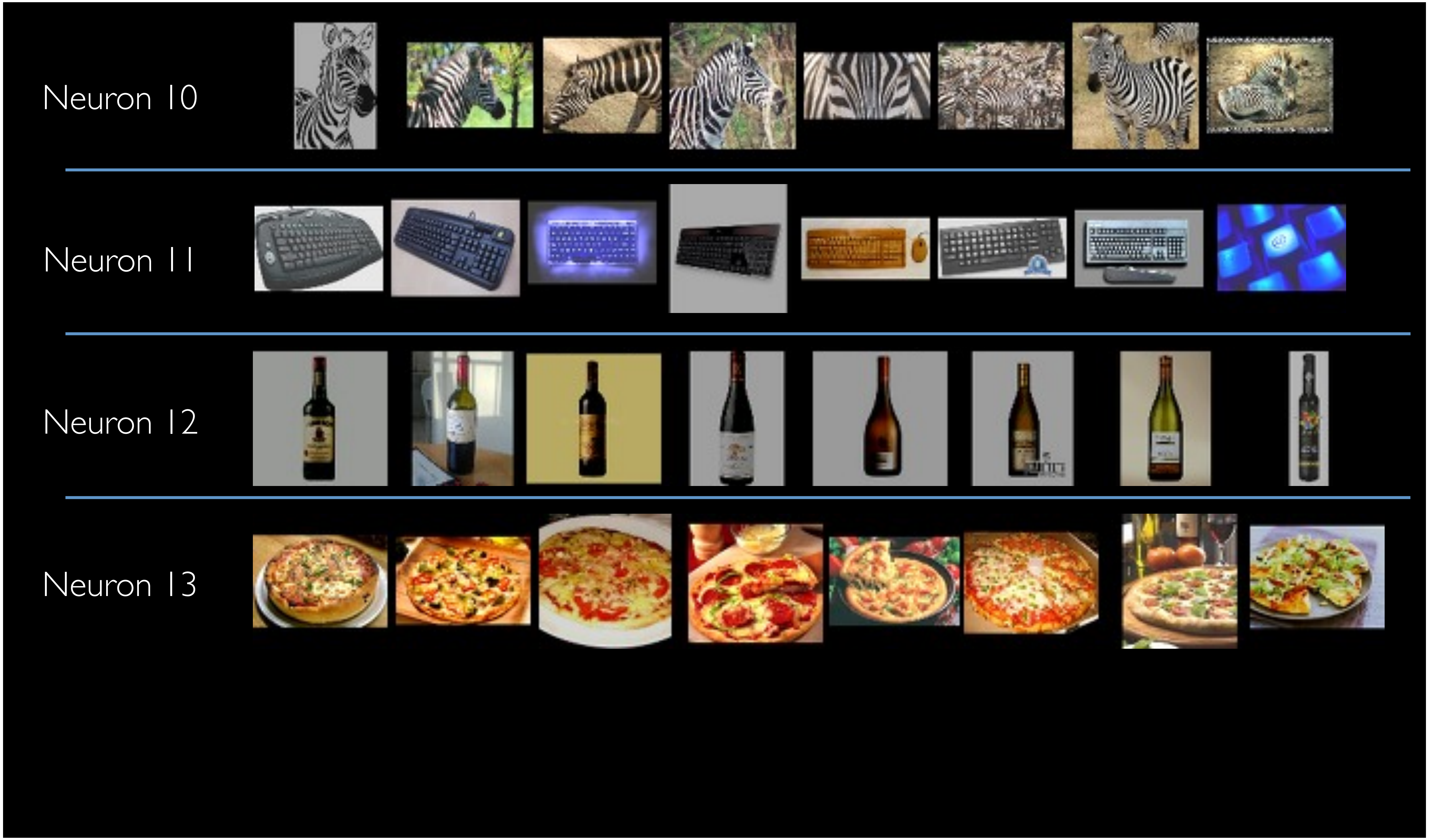
Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:

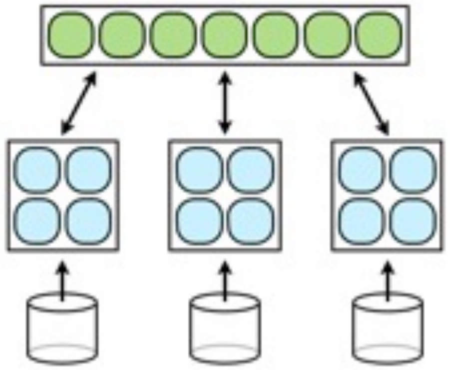


Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:



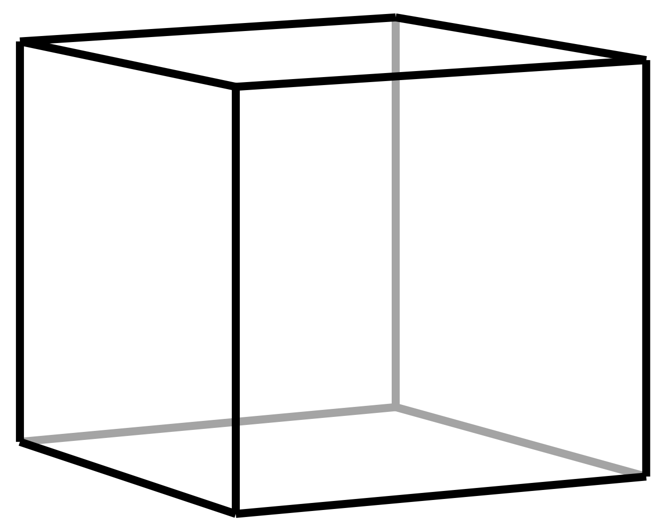
Applications



- Acoustic Models for Speech
- Unsupervised Feature Learning for Still Images
- Neural Language Models

Embeddings

~100-D joint embedding space

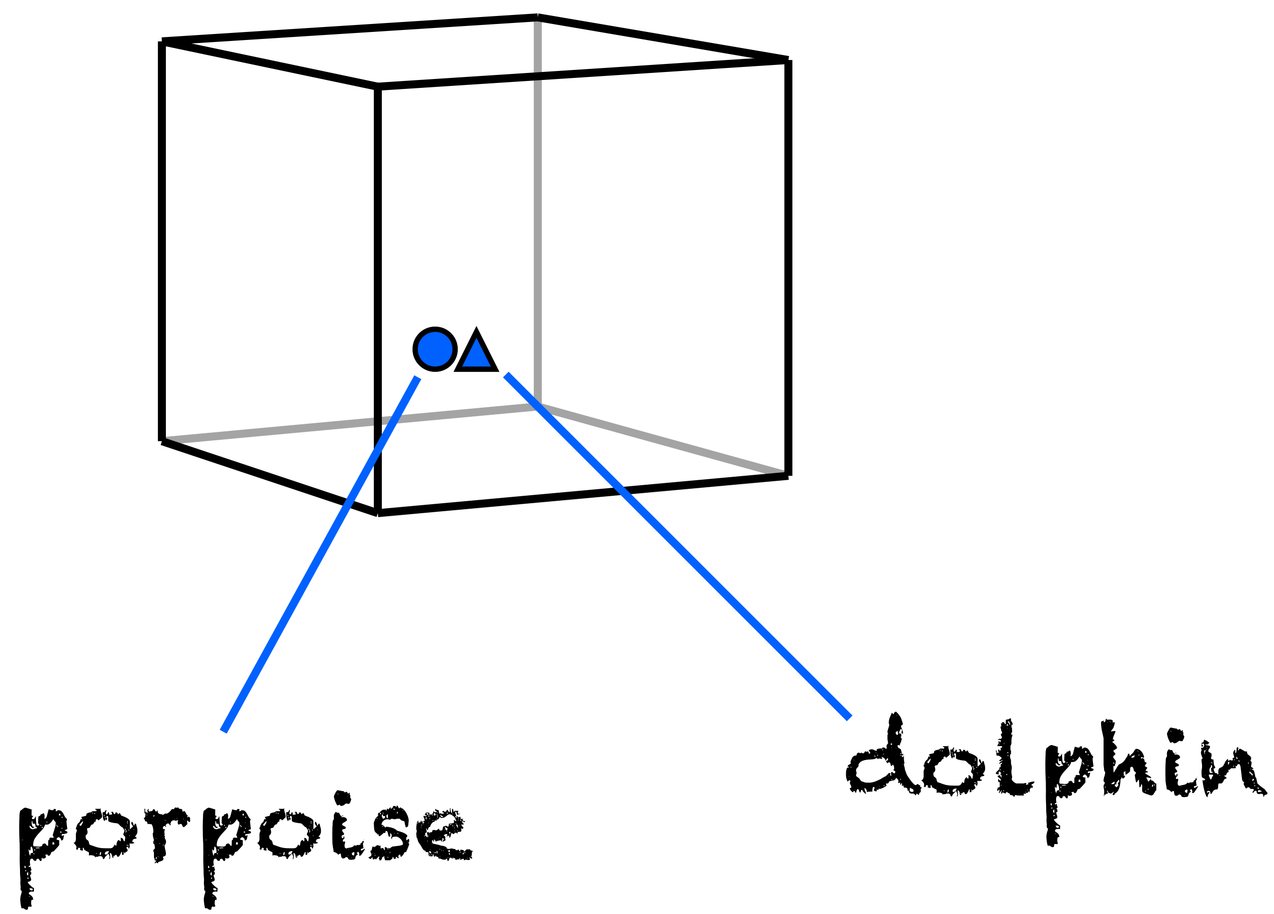


porpoise

dolphin

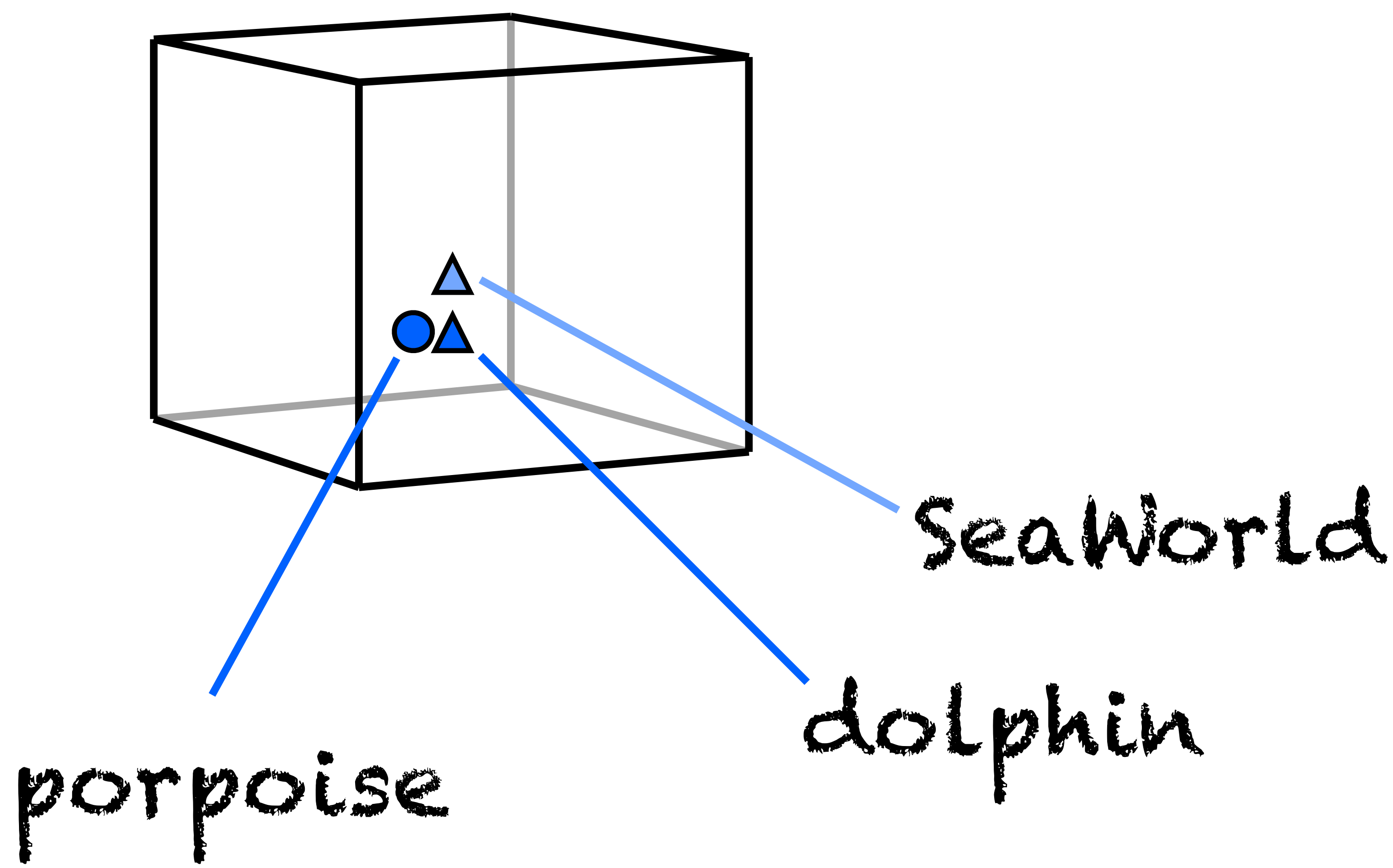
Embeddings

~100-D joint embedding space



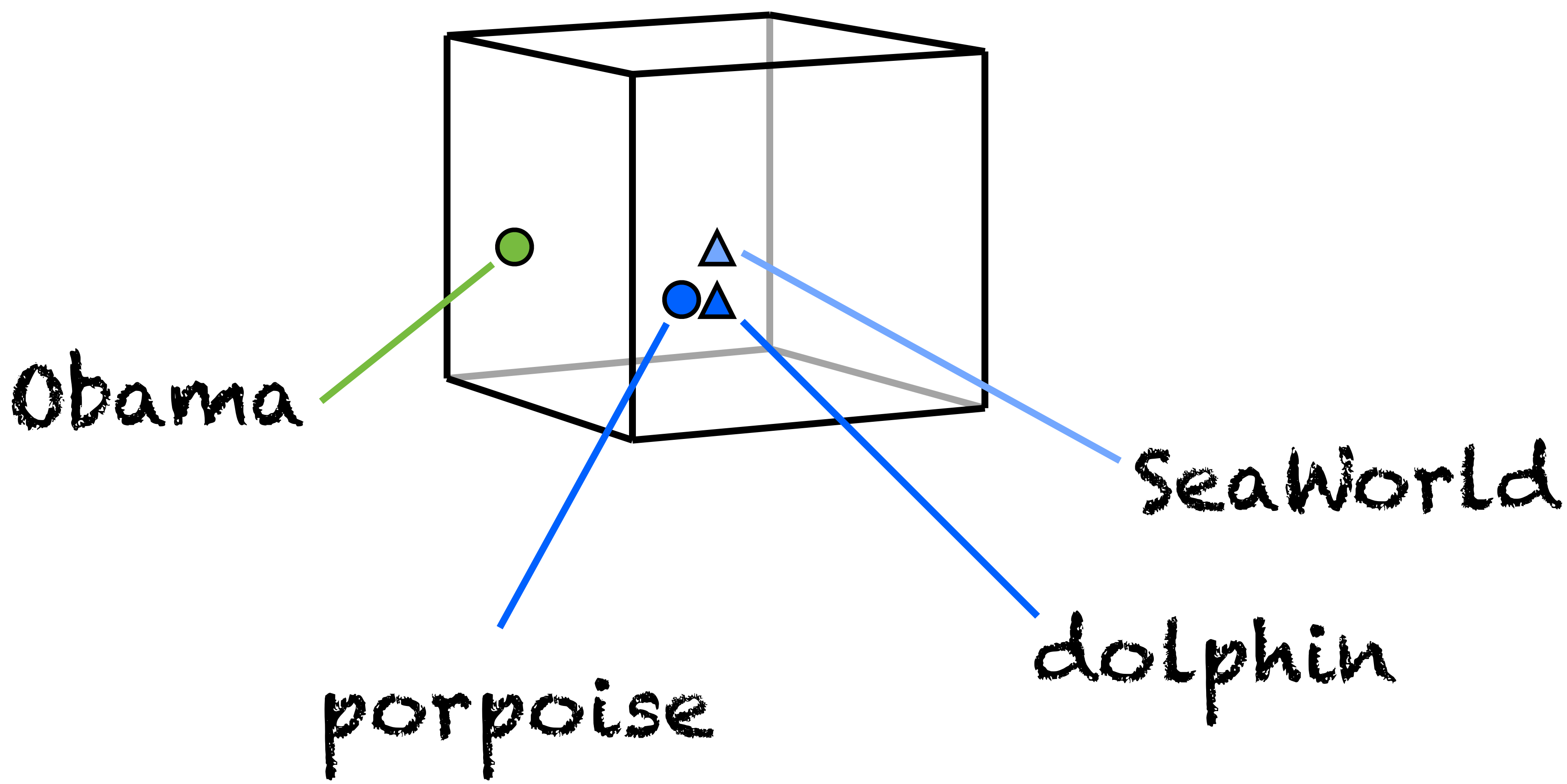
Embeddings

~100-D joint embedding space

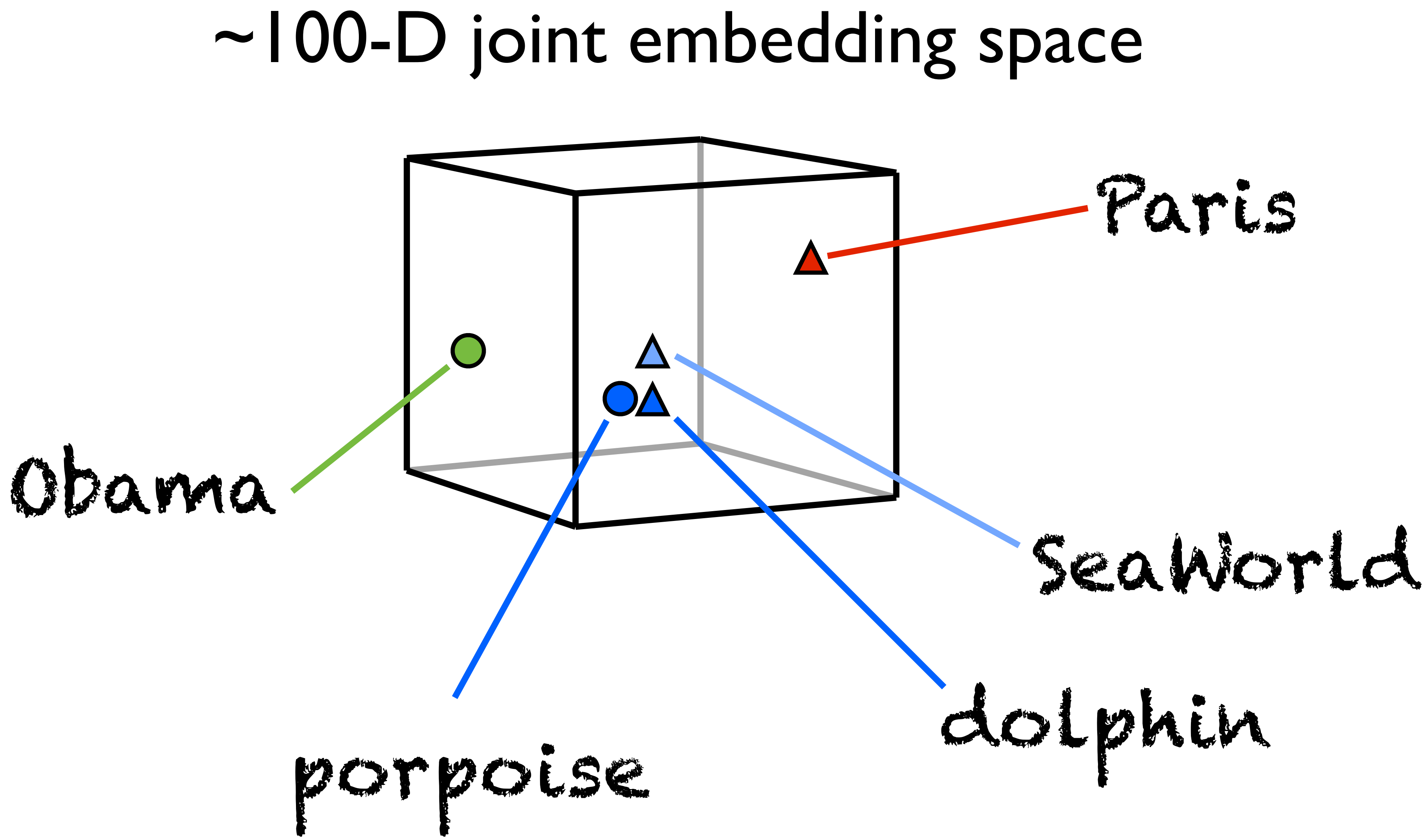


Embeddings

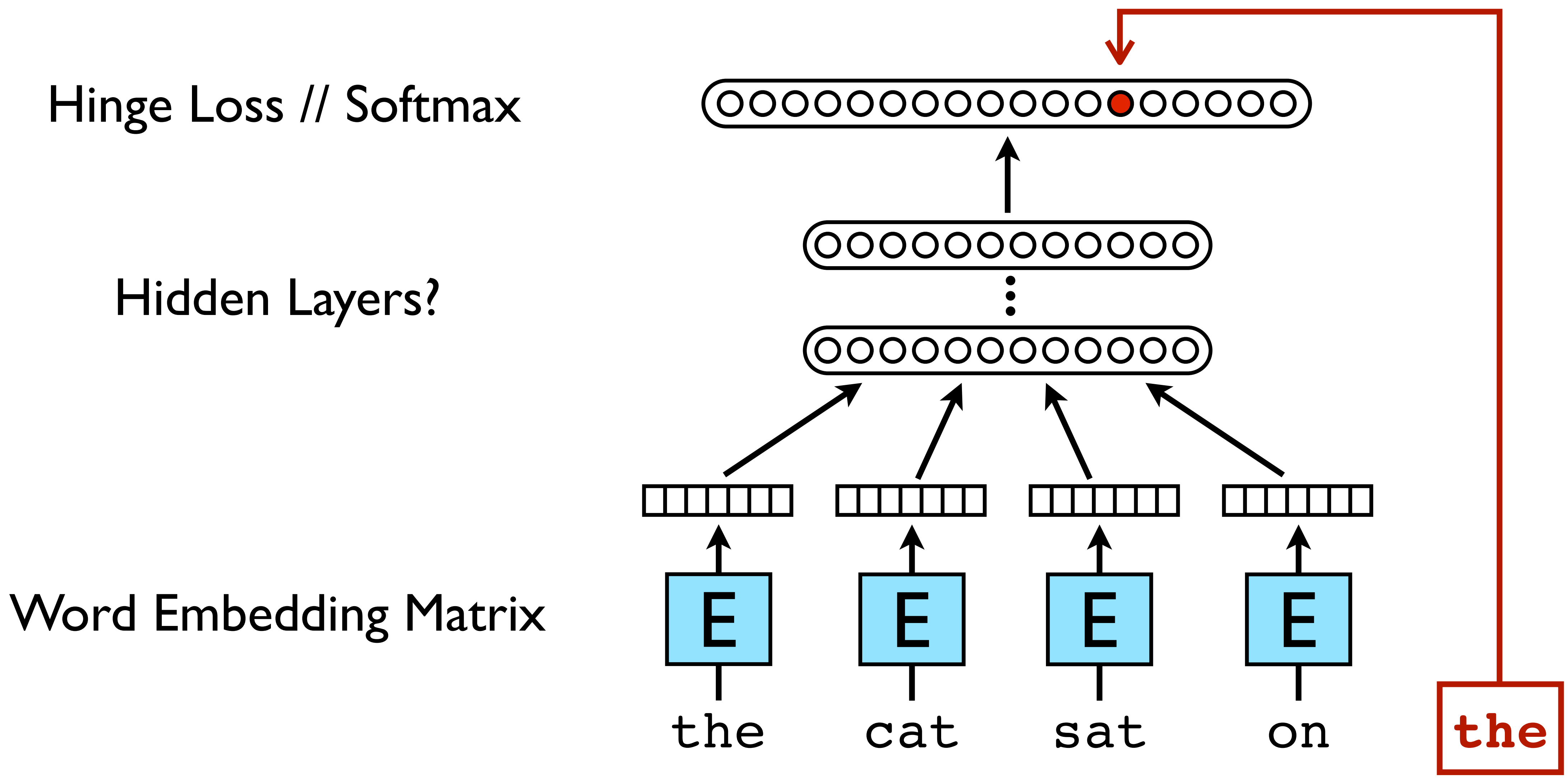
~100-D joint embedding space



Embeddings


~100-D joint embedding space
Paris
Obama
SeaWorld
dolphin
porpoise

Neural Language Models

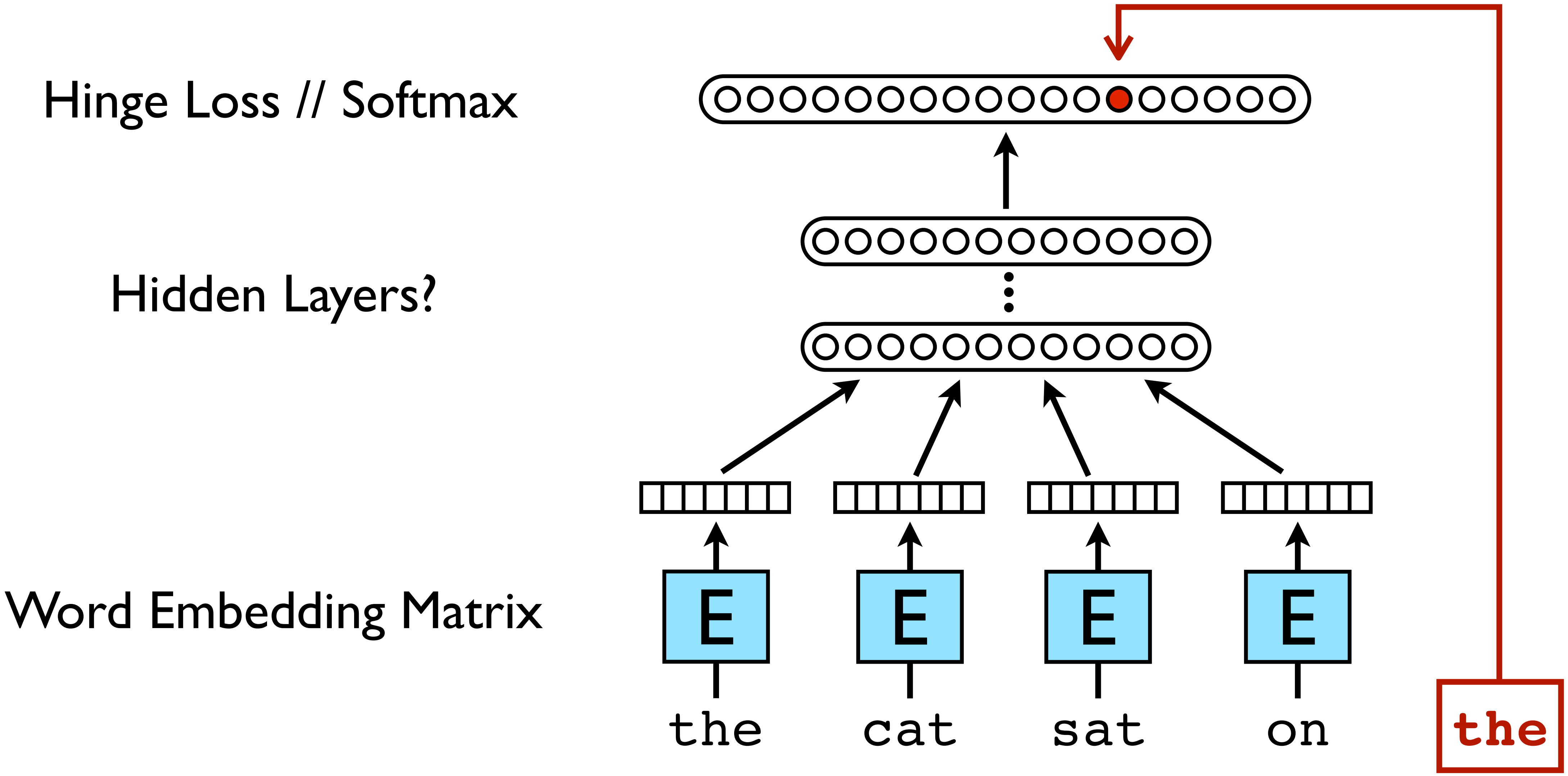


E is a matrix of dimension ||*Vocab*|| x *d*

Top prediction layer has ||*Vocab*|| x *h* parameters.

Most ideas from Bengio et al 2003, Collobert & Weston 2008

Neural Language Models



E is a matrix of dimension ||*Vocab*|| x *d*

Top prediction layer has ||*Vocab*|| x *h* parameters.

} 100s of millions of parameters, but gradients very sparse

Most ideas from Bengio et al 2003, Collobert & Weston 2008

Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

Cluster 1: apple

Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
11114	0.000000	Remove	apple
5026	0.652580	Add	fruit
14080	0.699192	Add	apples
48657	0.717818	Add	melon
28498	0.722390	Add	peach
39795	0.729893	Add	blueberry
35570	0.730500	Add	berry
25974	0.739561	Add	strawberry
46156	0.745343	Add	pecan
11907	0.756422	Add	potato
33847	0.759111	Add	pear
30895	0.763317	Add	mango
17848	0.768230	Add	pumpkin
39133	0.770143	Add	almond
14395	0.773105	Add	tomato
18163	0.782610	Add	onion
10470	0.782994	Add	pie
3023	0.787229	Add	tree
20340	0.793602	Add	bean
34968	0.794979	Add	watermelon

Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

Cluster 1: apple

Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
11114	0.000000	Remove	apple
5026	0.652580	Add	fruit
14080	0.699192	Add	apples
48657	0.717818	Add	melon
28498	0.722390	Add	peach
39795	0.729893	Add	blueberry
35570	0.730500	Add	berry
25974	0.739561	Add	strawberry
46156	0.745343	Add	pecan
11907	0.756422	Add	potato
33847	0.759111	Add	pear
30895	0.763317	Add	mango
17848	0.768230	Add	pumpkin
39133	0.770143	Add	almond
14395	0.773105	Add	tomato
18163	0.782610	Add	onion
10470	0.782994	Add	pie
3023	0.787229	Add	tree
20340	0.793602	Add	bean
34968	0.794979	Add	watermelon

Cluster 1: stab

Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
14979	0.000000	Remove	stab
7728	0.868853	Add	punch
469	0.909304	Add	shot
12820	0.909750	Add	thrust
8934	0.939908	Add	shell
10880	0.951466	Add	hammer
6975	0.951679	Add	bullet
1848	0.962053	Add	push
10888	0.962319	Add	eyed
718	0.965448	Add	hand
5865	0.966663	Add	grab
4611	0.967574	Add	swing
302	0.975696	Add	hit
869	0.976967	Add	force
1597	0.977625	Add	attempt
5977	0.978384	Add	finger
6162	0.978776	Add	knife
3434	0.980028	Add	sharp
1504	0.980160	Add	struck
39157	0.980219	Add	slug

Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

Cluster 1: apple

Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
11114	0.000000	Remove	apple
5026	0.652580	Add	fruit
14080	0.699192	Add	apples
48657	0.717818	Add	melon
28498	0.722390	Add	peach
39795	0.729893	Add	blueberry
35570	0.730500	Add	berry
25974	0.739561	Add	strawberry
46156	0.745343	Add	pecan
11907	0.756422	Add	potato
33847	0.759111	Add	pear
30895	0.763317	Add	mango
17848	0.768230	Add	pumpkin
39133	0.770143	Add	almond
14395	0.773105	Add	tomato
18163	0.782610	Add	onion
10470	0.782994	Add	pie
3023	0.787229	Add	tree
20340	0.793602	Add	bean
34968	0.794979	Add	watermelon

Cluster 1: stab

Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
14979	0.000000	Remove	stab
7728	0.868853	Add	punch
469	0.909304	Add	shot
12820	0.909750	Add	thrust
8934	0.939908	Add	shell
10880	0.951466	Add	hammer
6975	0.951679	Add	bullet
1848	0.962053	Add	push
10888	0.962319	Add	eyed
718	0.965448	Add	hand
5865	0.966663	Add	grab
4611	0.967574	Add	swing
302	0.975696	Add	hit
869	0.976967	Add	force
1597	0.977625	Add	attempt
5977	0.978384	Add	finger
6162	0.978776	Add	knife
3434	0.980028	Add	sharp
1504	0.980160	Add	struck
39157	0.980219	Add	slug

Cluster 1: iPhone

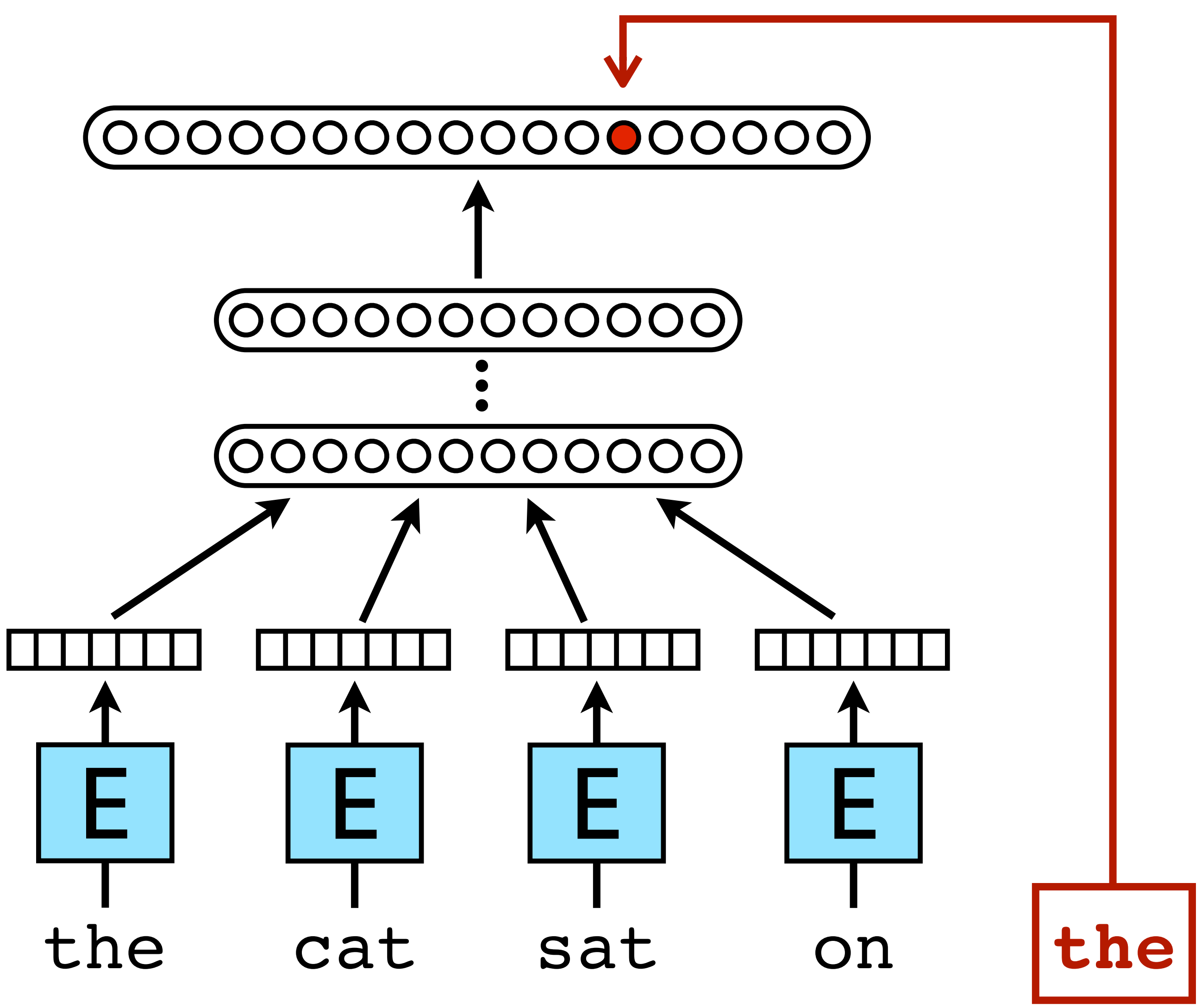
Cluster 1

Columns | Row filter (regexp)

Id	Distance↑	Adjust	Word
2964	0.000000	Remove	iPhone
6377	0.359153	Add	iPad
22542	0.554838	Add	iOS
10081	0.585379	Add	smartphone
5824	0.587948	Add	iPod
43921	0.608292	Add	PlayBook
18025	0.653021	Add	iPhones
6439	0.656983	Add	Android
38104	0.681779	Add	3GS
8088	0.690880	Add	BlackBerry
24581	0.696648	Add	Zune
33435	0.713150	Add	Smartphone
19186	0.714883	Add	Blackberry
9326	0.715027	Add	handset
26020	0.739856	Add	Droid
30557	0.756973	Add	Treo
12057	0.762164	Add	smartphones
6878	0.769016	Add	app
8211	0.779153	Add	iTunes
28120	0.787939	Add	iPads

Neural Language Models

- 7 Billion word Google News training set
- 1 Million word vocabulary
- 8 word history, 50 dimensional embedding
- Three hidden layers each w/200 nodes
- 50-100 asynchronous model workers

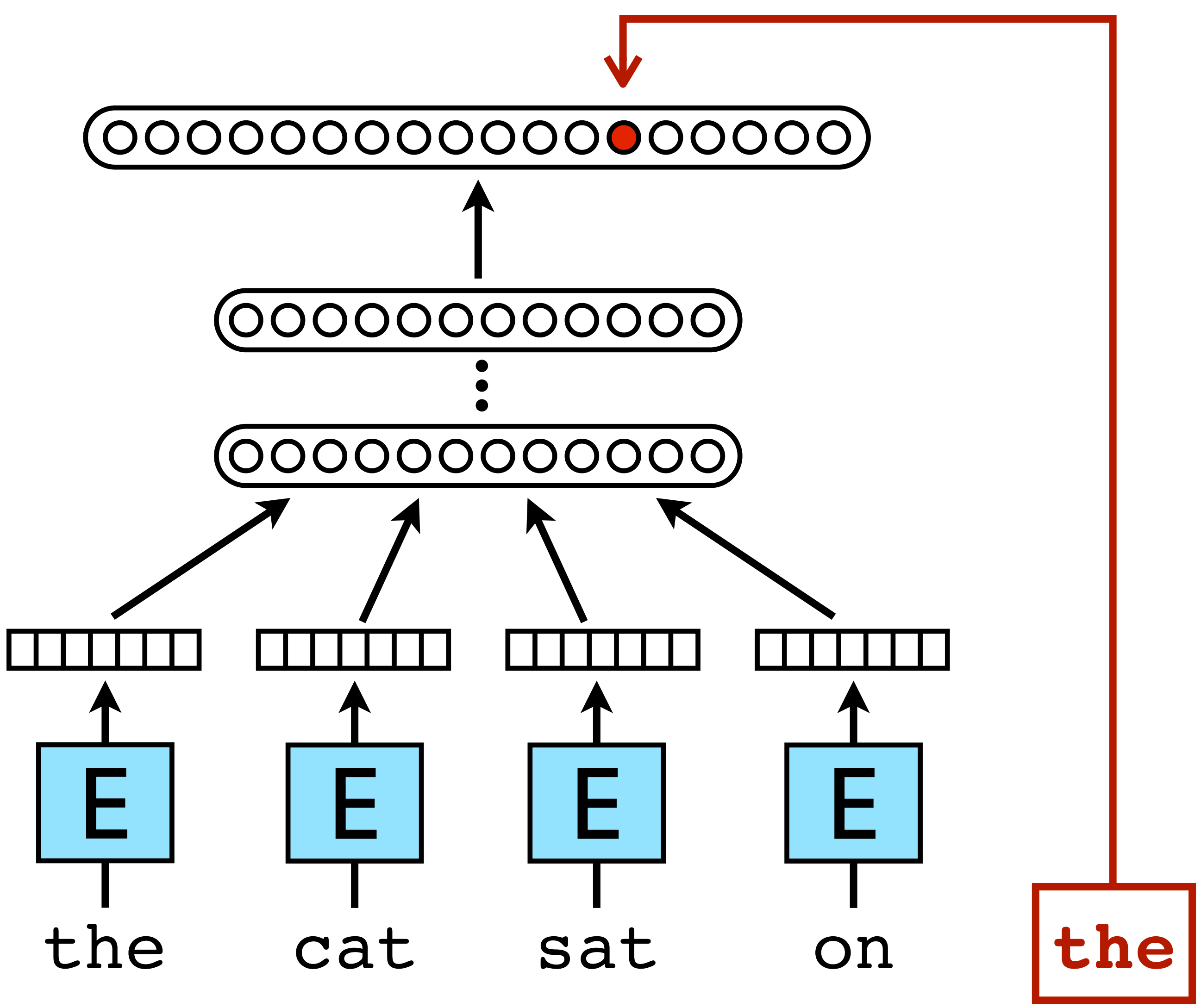


Neural Language Models

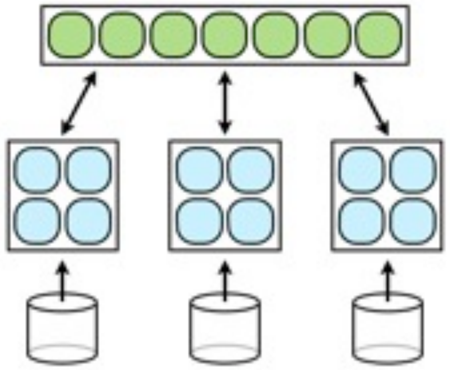
- 7 Billion word Google News training set
- 1 Million word vocabulary
- 8 word history, 50 dimensional embedding
- Three hidden layers each w/200 nodes
- 50-100 asynchronous model workers

Perplexity Scores

Traditional 5-gram	XXX
NLM	+15%
5-gram + NLM	-33%



Deep Learning Applications



Many other applications not discussed today:

- Clickthrough prediction for advertising
- Video understanding
- User action prediction

...

Thanks! Questions...?

Further reading:

- Ghemawat, Gobioff, & Leung. *Google File System*, SOSP 2003.
- Barroso, Dean, & Hölzle. *Web Search for a Planet:The Google Cluster Architecture*, IEEE Micro, 2003.
- Dean & Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, OSDI 2004.
- Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, & Gruber. *Bigtable:A Distributed Storage System for Structured Data*, OSDI 2006.
- Brants, Popat, Xu, Och, & Dean. *Large Language Models in Machine Translation*, EMNLP 2007.
- Le, Ranzato, Monga, Devin, Chen, Corrado, Dean, & Ng. *Building High-Level Features Using Large Scale Unsupervised Learning*, ICML 2012.
- Dean et al. , *Large Scale Distributed Deep Networks,* to appear NIPS 2012.
- Corbett, Dean, ... Ghemawat, et al. *Spanner: Google's Globally-Distributed Database*, to appear in OSDI 2012
- Dean & Barroso, *The Tail at Scale*, to appear in CACM 2012/2013.

- Protocol Buffers. http://code.google.com/p/protobuf/
- Snappy. http://code.google.com/p/snappy/
- Google Perf Tools. http://code.google.com/p/google-perftools/
- LevelDB. http://code.google.com/p/leveldb/

These and many more available at: http://labs.google.com/papers.html