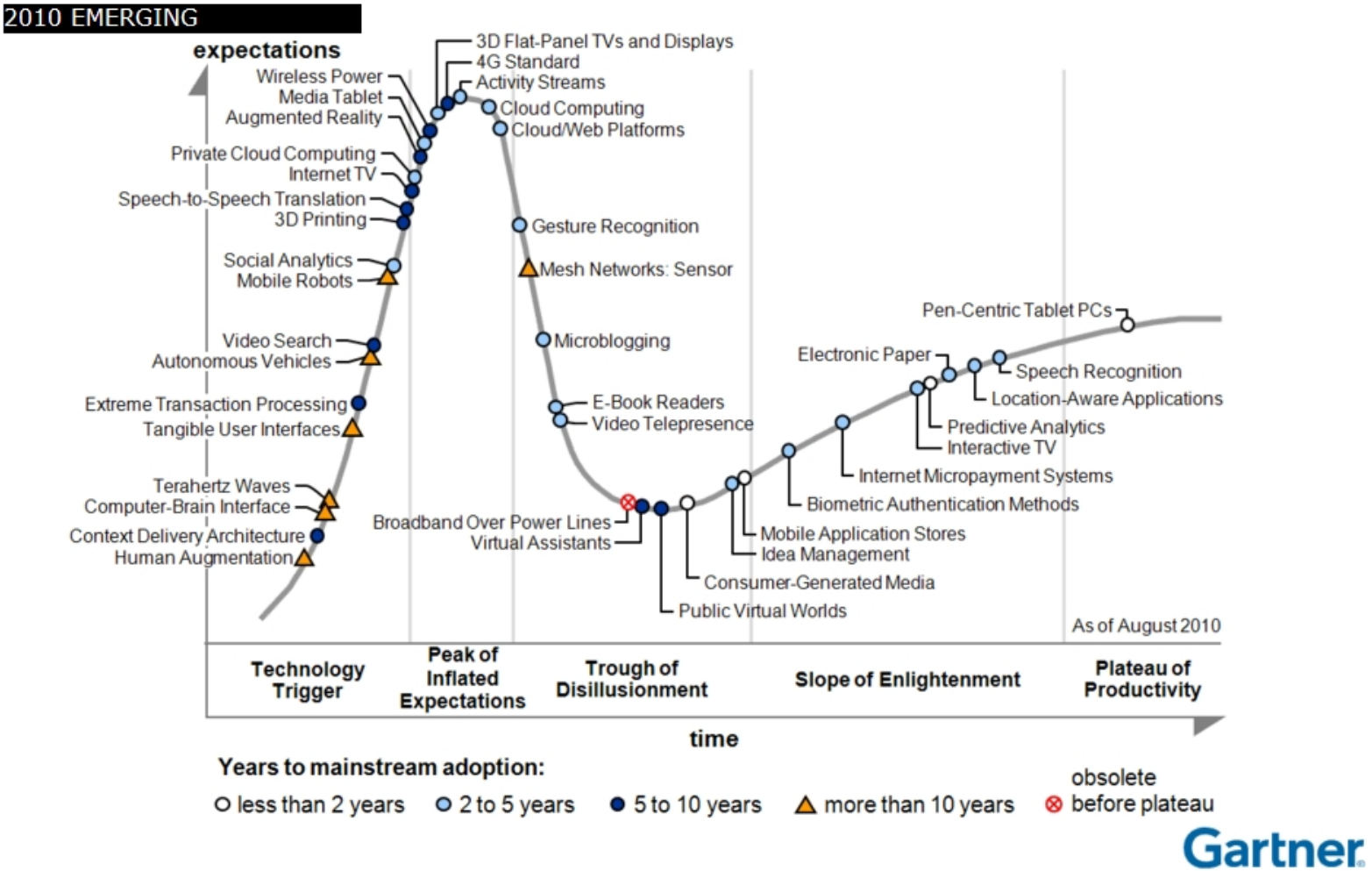
Introducing the Cloud

Gwendal Simon

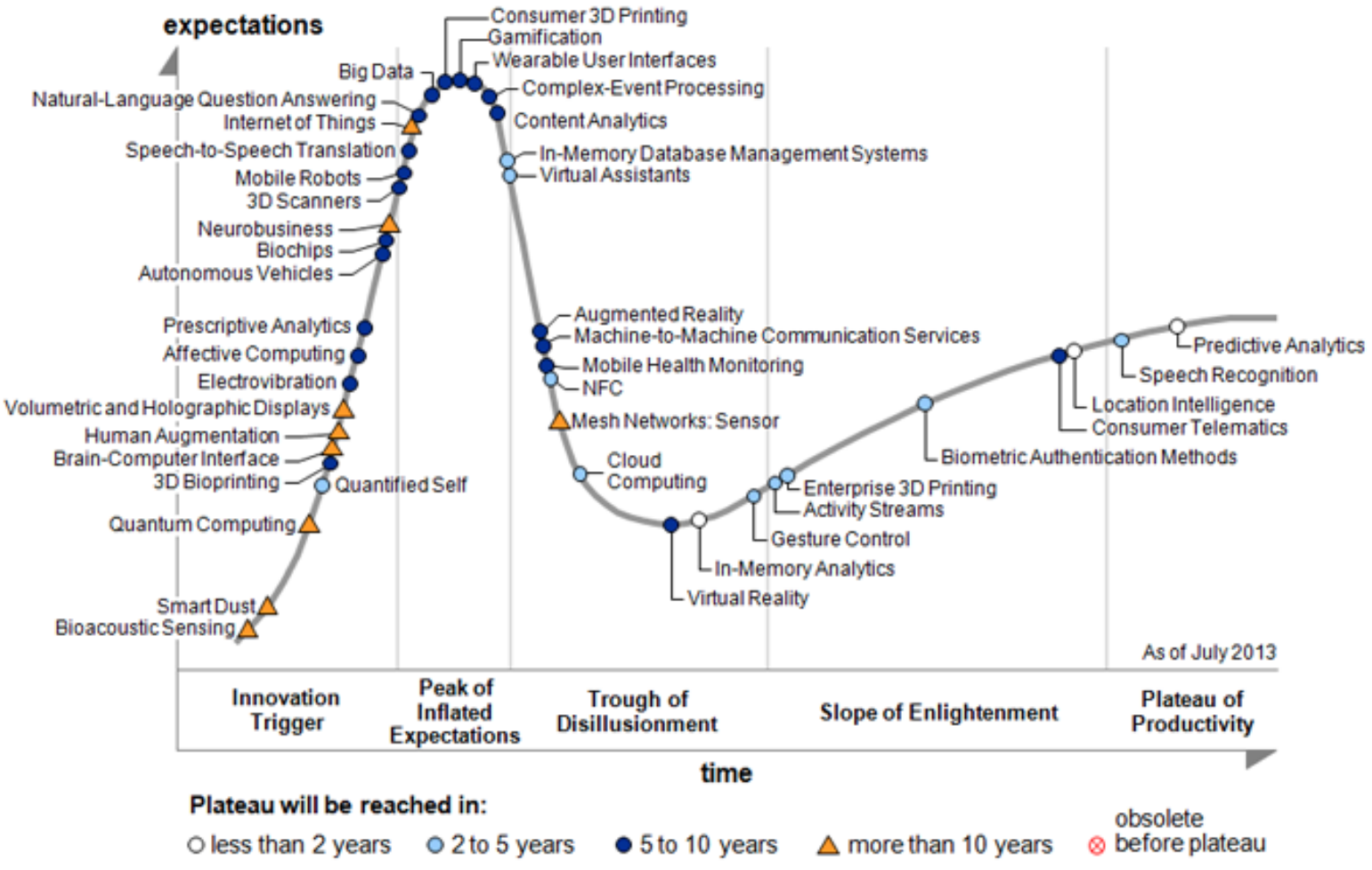
http://perso.telecom-bretagne.eu/gwendalsimon
@gwendal

Already hype when you entered TB



Gartner

Hype in a different way now



Avancement

In a nutshell

In a nutshell

Cloud computing is a model for enabling convenient, *on-demand* network access to a *shared* pool of configurable computing resources (*e.g.*, networks, servers, storage, applications, and services) that can be rapidly *provisioned* and released with *minimal management effort* or service provider interaction

NIST : http ://csrc.nist.gov/groups/SNS/cloud-computing/index.html

The *aaS paradigm

The *aaS paradigm

- SaaS : Software as a Service
  - applications for *end-users*
    - email, office suite, photos sharing, video storage

The ∗aaS paradigm

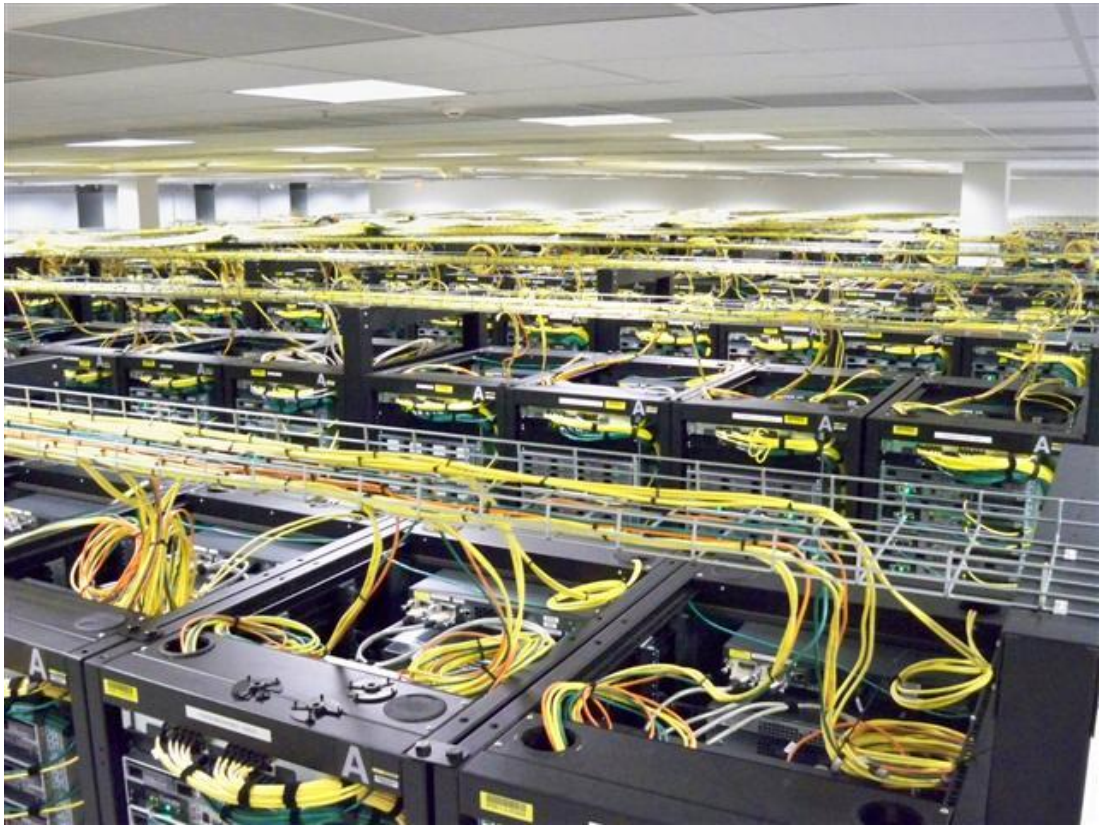
- SaaS : Software as a Service
  - applications for *end-users*
    - email, office suite, photos sharing, video storage
- PaaS : Platform as a Service
  - services for *web app developers*
    - workflow facilities and various basic services

The ∗aaS paradigm

- SaaS : Software as a Service
  - applications for *end-users*
    - email, office suite, photos sharing, video storage
- PaaS : Platform as a Service
  - services for *web app developers*
    - workflow facilities and various basic services
- IaaS : Infrastructure as a Service
  - resources for *developers*
    - servers, network equipment, memory, CPU

Selected Pictures

Selected Pictures



Selected Pictures

Selected Pictures

Avancement

Case Studies

- A Commercial Offer : Amazon
- Switching to the Cloud : Netflix

Amazon Web Services

- DynamoDB
- Simple Storage Service (S3)
- Elastic Compute Cloud (EC2)
- Elastic Load Balancing (ELB)
- Virtual Private Cloud (VPC)
- Glacier
- Elastic MapReduce (EMR)
- CloudFront

Case Studies

- A Commercial Offer : Amazon
- Switching to the Cloud : Netflix

See Slides

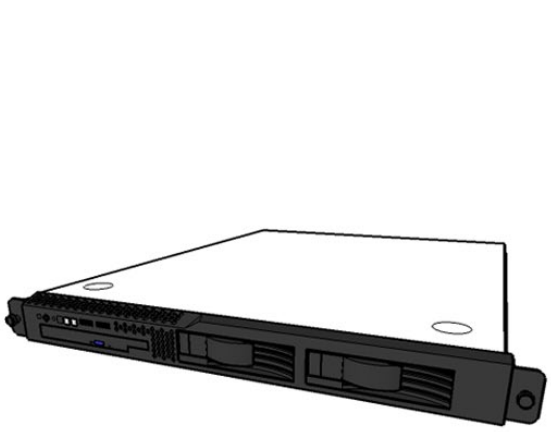
http ://www.slideshare.net/adrianco/netflix-global-cloud

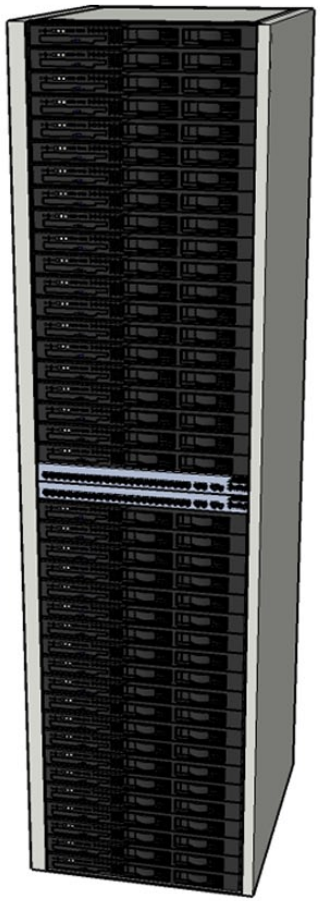
Avancement

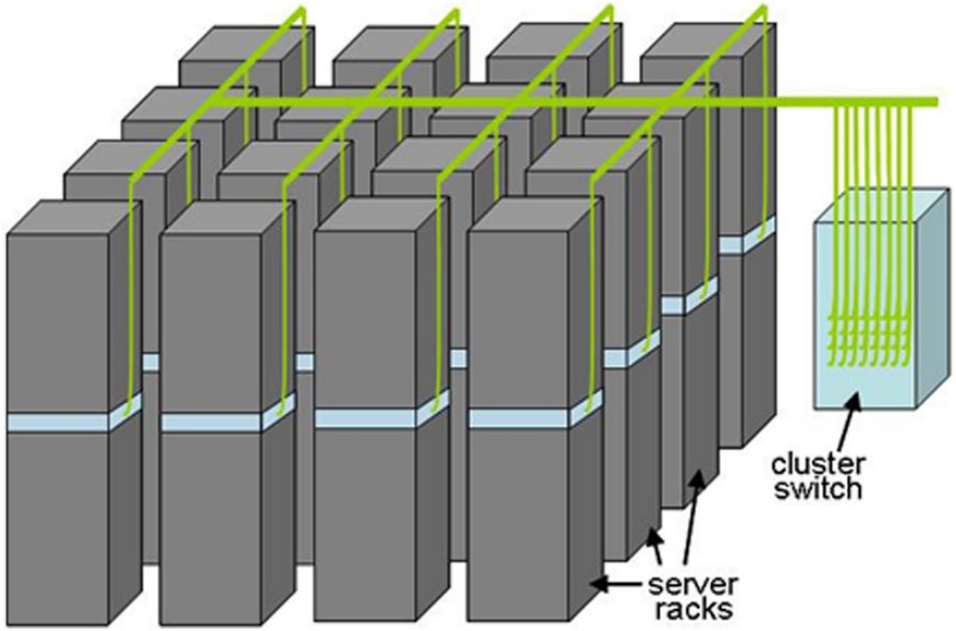
Basic Elements of a Datacenter

- Computing Architecture
- Energy
- Dealing with Failures

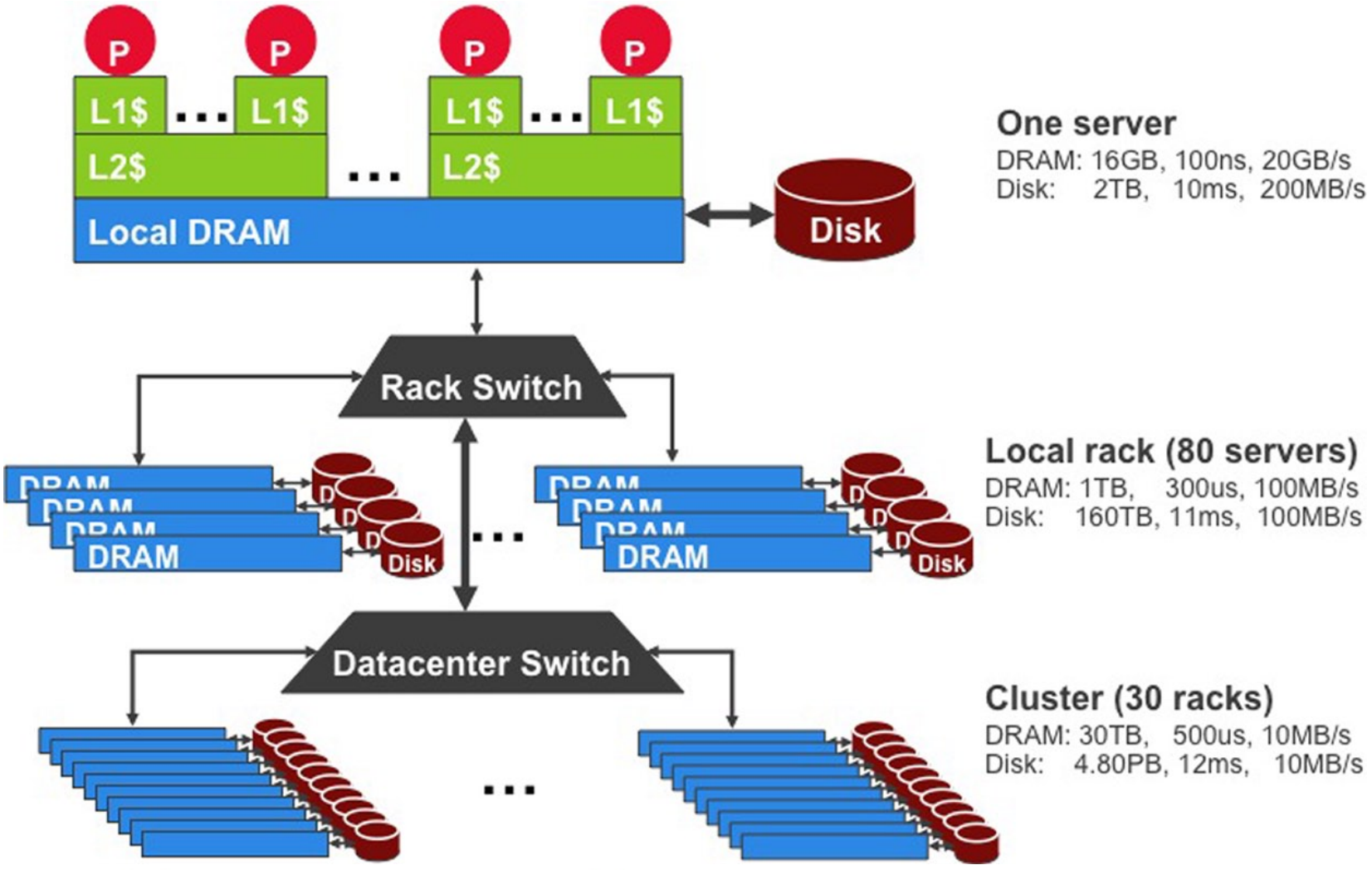
Architecture Basics







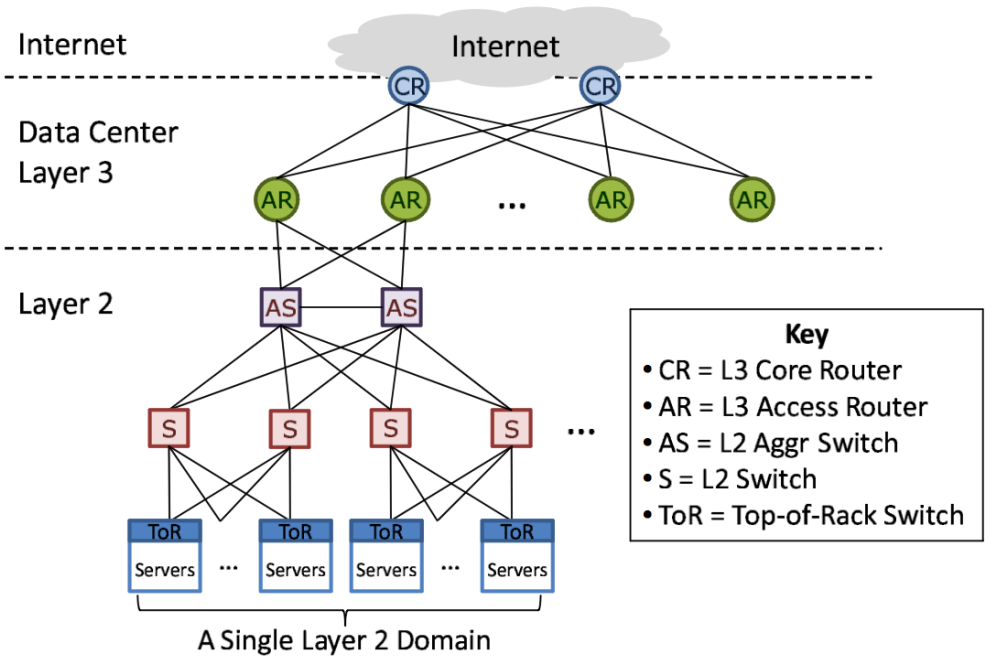
Storage and Access Time (in 2009)



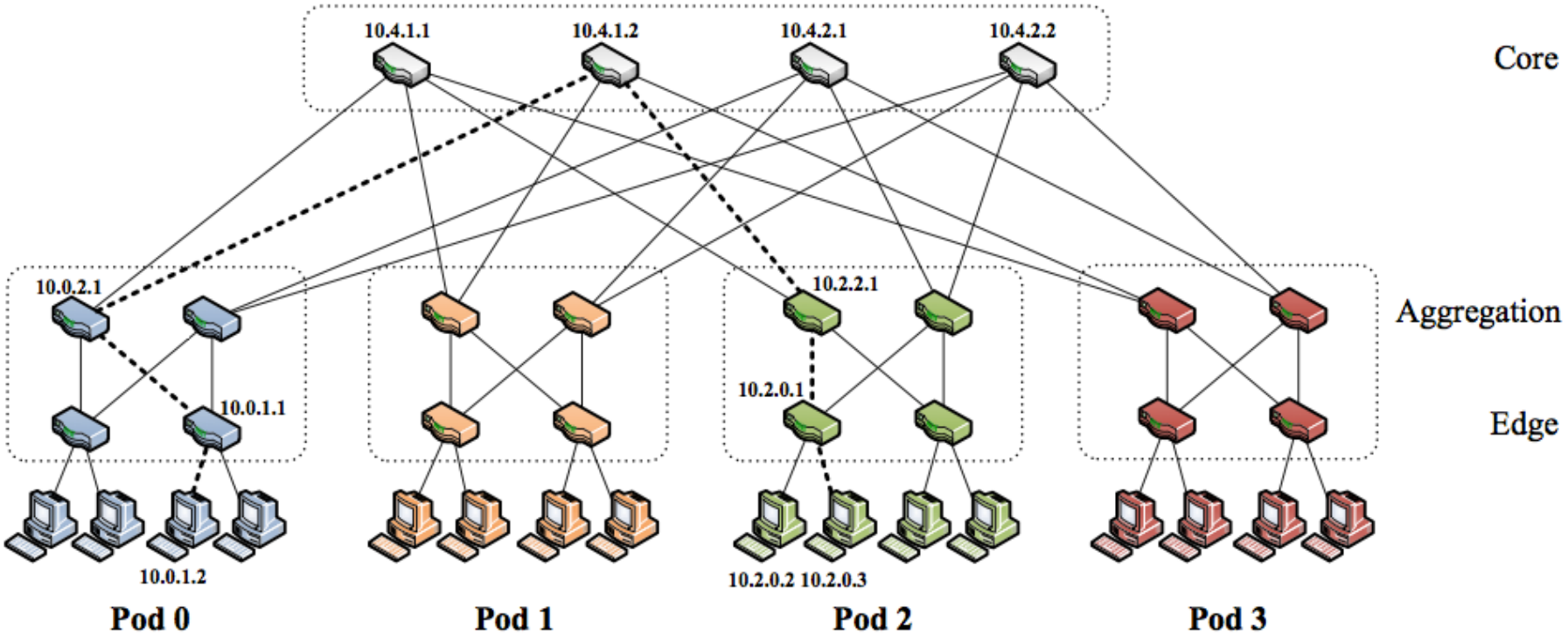
Network

Oversubscription :

$$\frac{\text{worst-case communication between servers}}{\text{capacity of network topology}}$$



Network Fat-Tree Topologies



Typical Technologies and Prices (in 2011)

Server :

- computation : 24-32 cores at $200 per core
- network : $150 per 10 Gbps port NIC

Typical Technologies and Prices (in 2011)

Server :

- computation : 24-32 cores at $200 per core
- network : $150 per 10 Gbps port NIC

10 Gbps switch :

- today : 48-64 ports at $450 per port
- trend : up to 150 ports at $100 per port

Typical Technologies and Prices (in 2011)

Server :

- computation : 24-32 cores at $200 per core
- network : $150 per 10 Gbps port NIC

10 Gbps switch :

- today : 48-64 ports at $450 per port
- trend : up to 150 ports at $100 per port

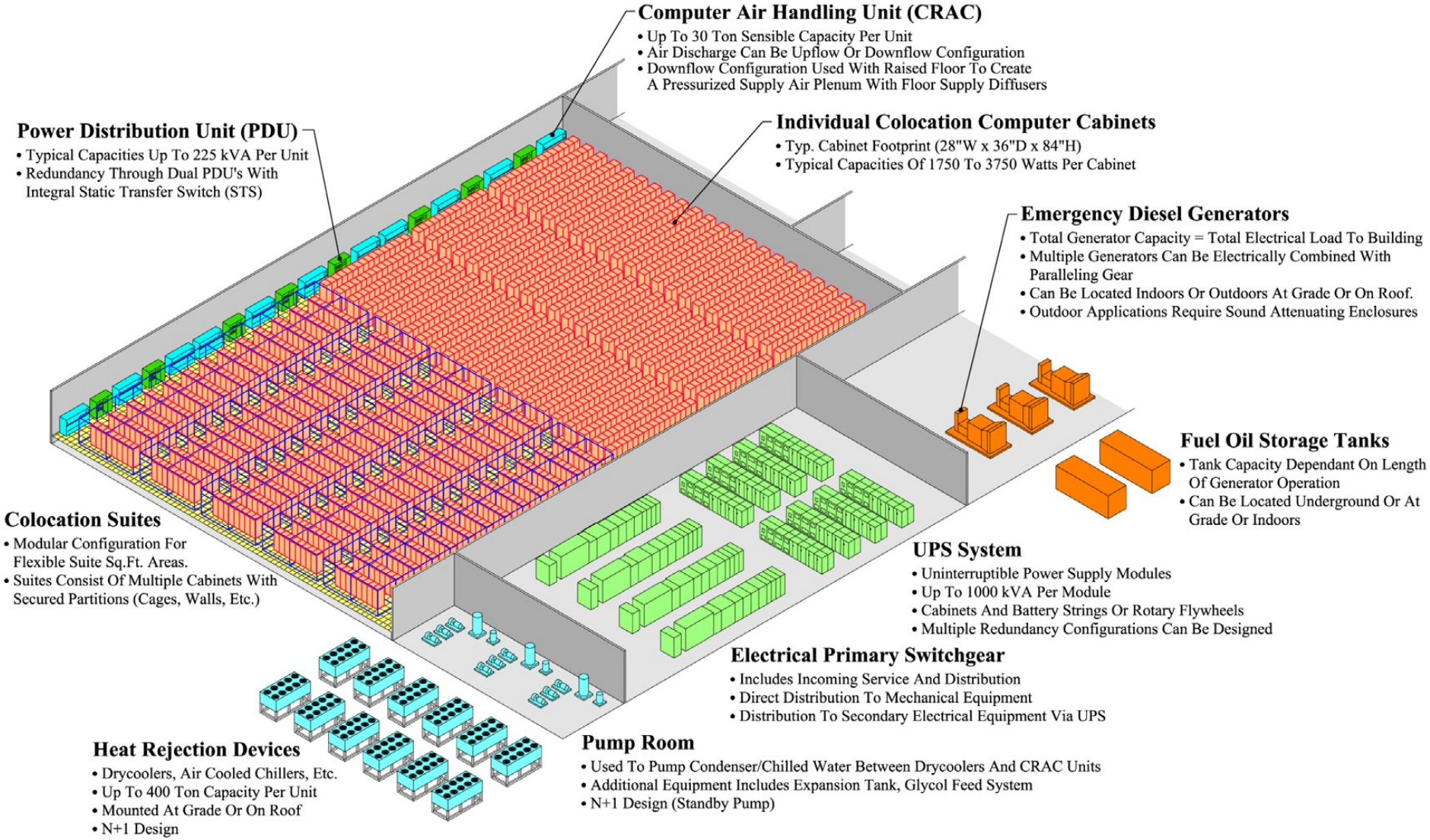
Memory :

Technology	Access time (ns)	Cost ($/MB)	Max. size
SRAM	4	27	$\approx$ 250 Mbits
RLDRAM	15	0.27	$\approx$ 2 Gbits
DRAM	55	0.016	$\approx$ 100 Gbits

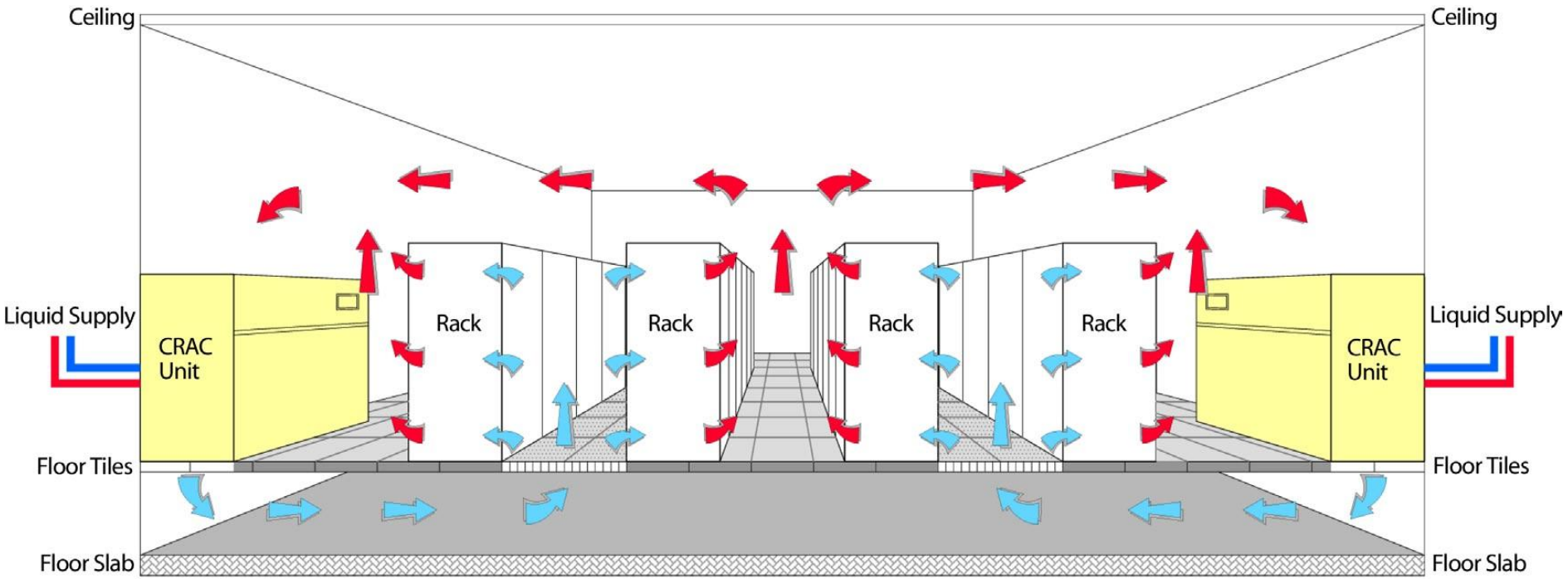
Basic Elements of a Datacenter

- Computing Architecture
- Energy
- Dealing with Failures

Main Electric Components



Cooling Challenge



Evaluating Energy Efficiency

$$\textit{Power Usage Effectiveness} = \frac{\text{total energy}}{\text{energy in equipment}}$$

- first generation datacenter PUE ≥ 3.0
- toward PUE around 1.1

Evaluating Energy Efficiency

$$\textit{Power Usage Effectiveness} = \frac{\text{total energy}}{\text{energy in equipment}}$$

- first generation datacenter PUE ≥ 3.0
- toward PUE around 1.1

$$\textit{Server PUE} = \frac{\text{total server power}}{\text{critical component power}}$$

- basic SPUE is 1.7
- state-of-the-art servers reach 1.1

Evaluating Energy Efficiency

$$\textit{Power Usage Effectiveness} = \frac{\text{total energy}}{\text{energy in equipment}}$$

- first generation datacenter PUE ≥ 3.0
- toward PUE around 1.1

$$\textit{Server PUE} = \frac{\text{total server power}}{\text{critical component power}}$$

- basic SPUE is 1.7
- state-of-the-art servers reach 1.1

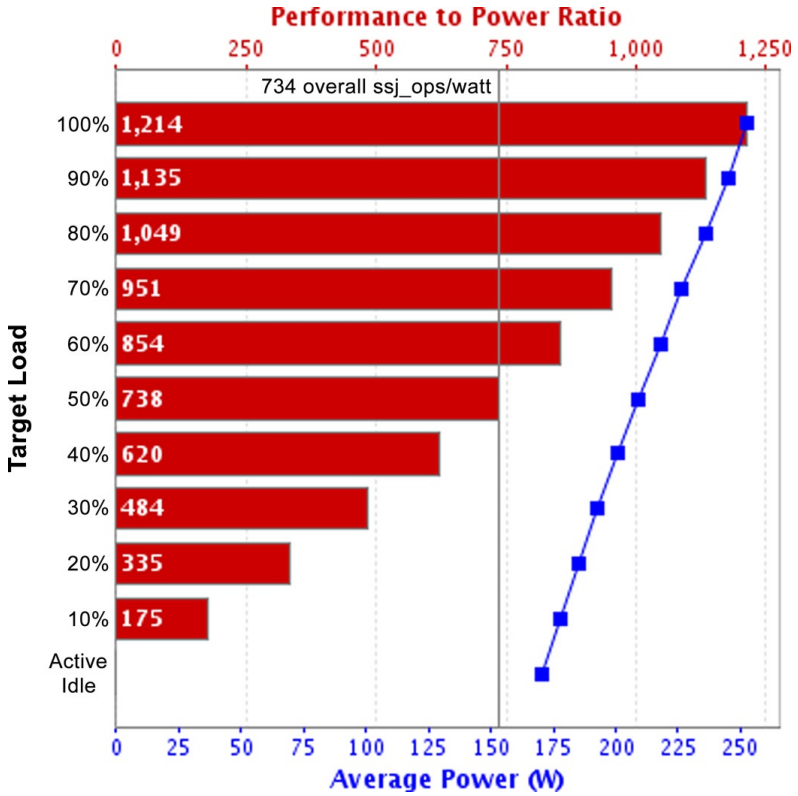
and **computing efficiency** ?

Computing Efficiency

Benchmarking cluster-level efficiency : on-going work

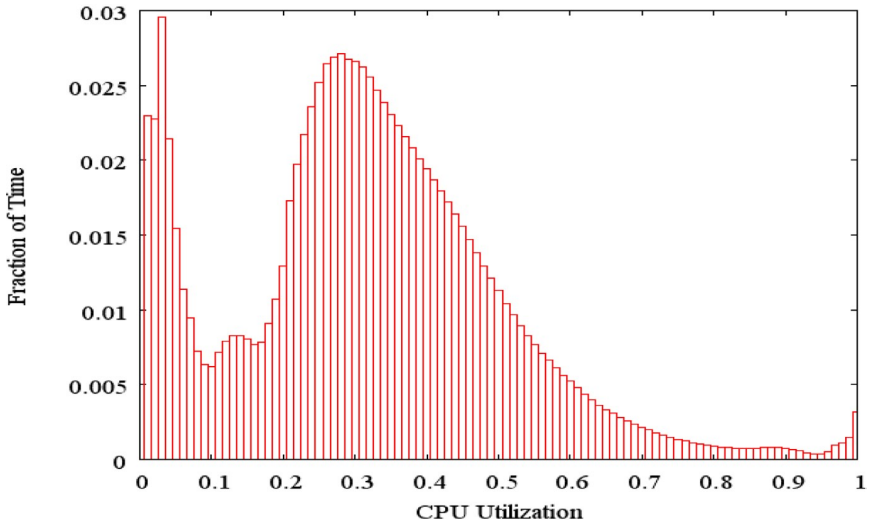
Computing Efficiency

Benchmarking cluster-level efficiency : on-going work
Benchmarking individual computer is easier

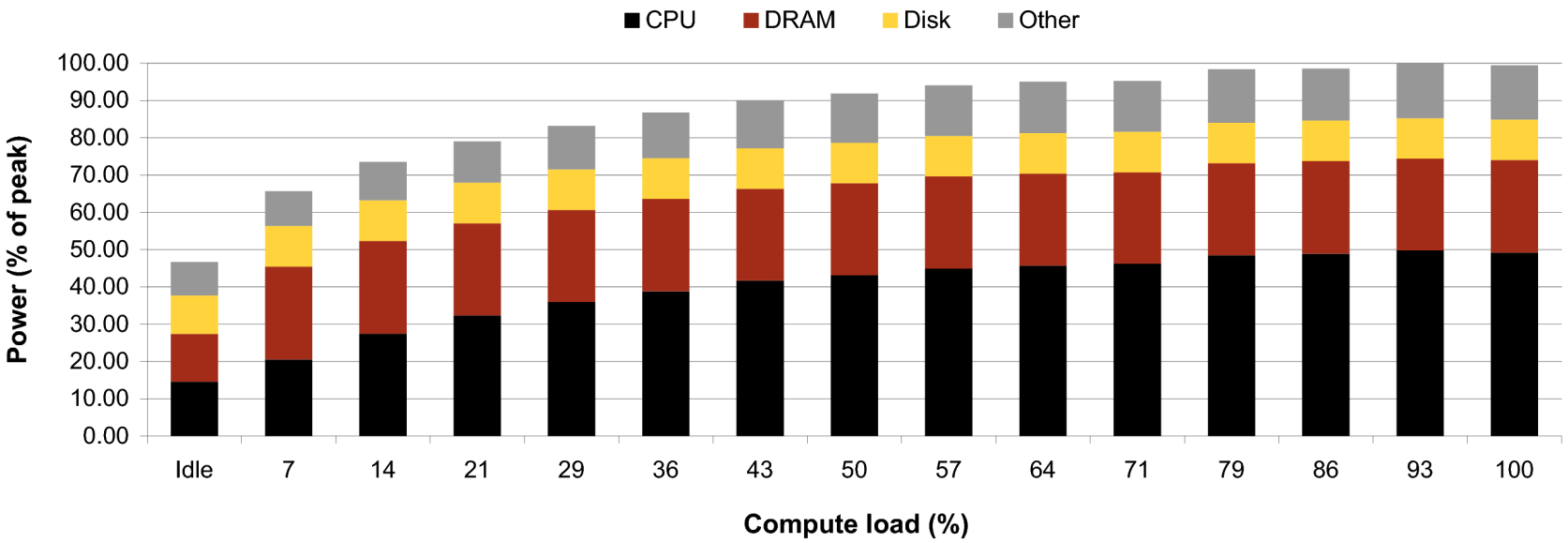


Computing Efficiency

Benchmarking cluster-level efficiency : on-going work
Benchmarking individual computer is easier



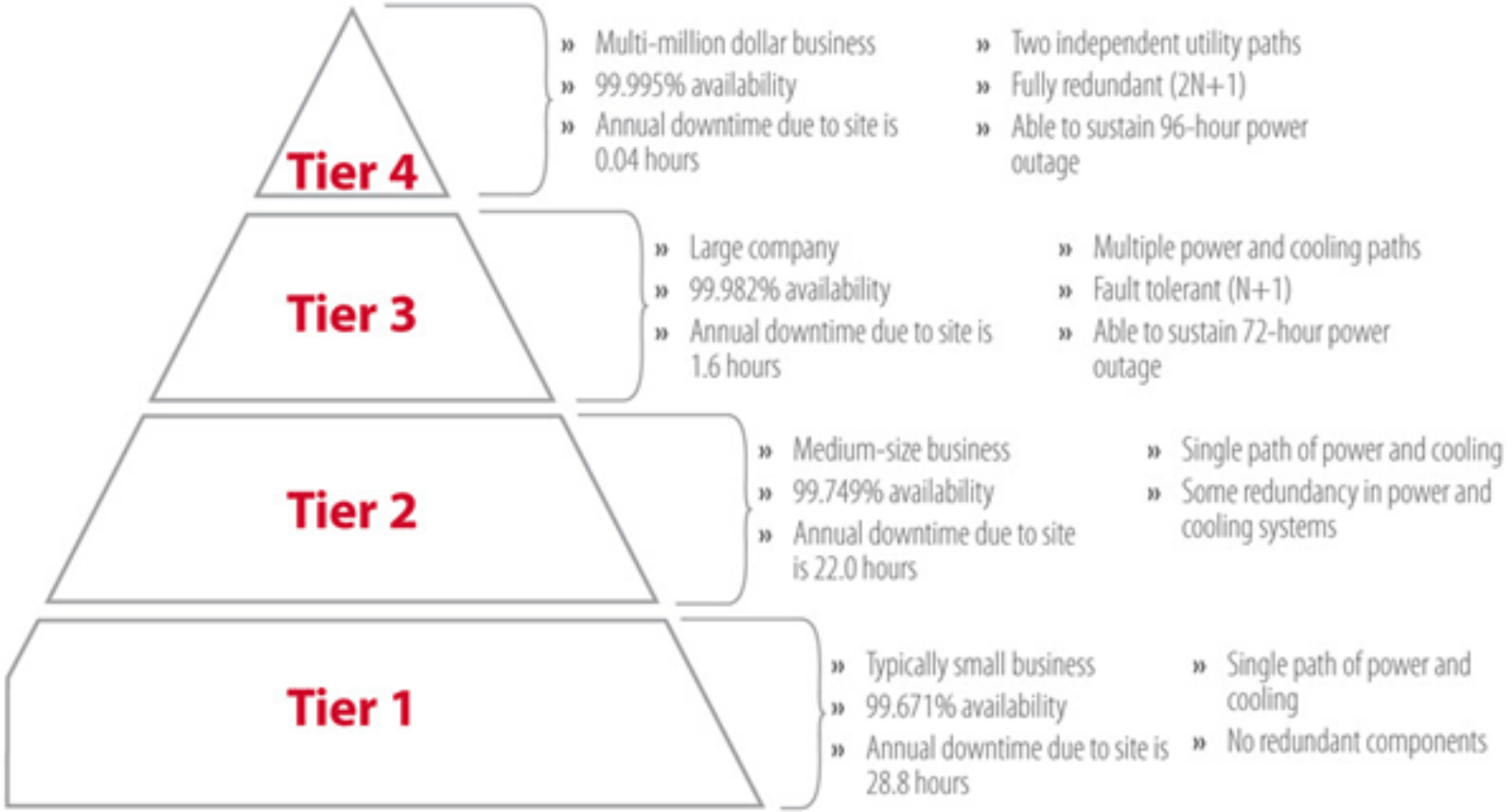
Toward Energy-Proportional Servers



Basic Elements of a Datacenter

- Computing Architecture
- Energy
- Dealing with Failures

Data-Center Tiers



Origin of Impacting Failures

Cause	% of events
software	33 %
configuration	28 %
human	13 %
network	12 %
hardware	11 %
other	3 %

Origin of Impacting Failures

Cause	% of events
software	33 %
configuration	28 %
human	13 %
network	12 %
hardware	11 %
other	3 %

hardware faults are masked by fault-tolerant software

Failures and Crash

Average machine availability is $\simeq 99.9\%$

- 95% of machines restart less than once a month
- 80% of restart events last less than 10 minutes

Failures and Crash

Average machine availability is $\simeq$ 99.9%

- 95% of machines restart less than once a month
- 80% of restart events last less than 10 minutes

Software most frequent faults (in one year) :

- *DRAM soft-errors* : 1% experience uncorrectable err
- *disk soft-errors* : 3% of drives see corrupted sectors

Avancement

Software Infrastructure

- Cluster-Level : MapReduce

Motivation

Map/Reduce is a software framework for easily writing applications which ***process*** *vast amounts of data* (multi-terabyte data-sets) *in-parallel* on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

Functional Programming

Two fundamentals functions :

- **map** : apply a function to a list of elements
  - `map f [] = []`
    `| map f [x::xs] = (f x) :: (map f xs)`
  - *map square [1,2,5]* $\rightarrow$ *[1,4,25]*

Functional Programming

Two fundamentals functions :

- **map** : apply a function to a list of elements
  - `map f [] = []`
    `| map f [x::xs] = (f x) :: (map f xs)`
  - *map square [1,2,5]* $\rightarrow$ *[1,4,25]*
- **reduce** : build a value from a function and a list
  - `reduce f a [] = a`
    `| reduce f a [x::xs] = reduce f (f x a) xs`
  - *reduce add 0 [1,3,6]* $\rightarrow$ *10*

MapReduce

Implementing two functions w.r.t data `(key,val)`

- **map** : smaller sub-problems distributed to nodes
  - `map (inKey, inVal)` $\rightarrow$ `list (outKey, v)`
  - produces *intermediate* values with an output key

MapReduce

Implementing two functions w.r.t data `(key,val)`

- **map** : smaller sub-problems distributed to nodes
  - `map (inKey, inVal)` $\rightarrow$ `list (outKey, v)`
  - produces *intermediate* values with an output key

- **reduce** : combines results of sub-problems
  - `reduce (outKey, list v)` $\rightarrow$ `outVal`
  - produces an output value from intermediate values

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)
```

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)

reduce(word, partCount):
  int result = 0
  for pc in partCount:
     result += pc
  emit(result)
```

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)

reduce(word, partCount):
  int result = 0
  for pc in partCount:
     result += pc
  emit(result)
```

map(file1, “hello me, goodbye me”)→
<hello,1> <me,1> <goodbye,1> <me,1>

map(file2, “hello you, bye you”)→
<hello,1> <you,1> <bye,1> <you,1>

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)

reduce(word, partCount):
  int result = 0
  for pc in partCount:
     result += pc
  emit(result)
```

map(file1, “hello me, goodbye me”)→
<hello,1> <me,1> <goodbye,1> <me,1>

map(file2, “hello you, bye you”)→
<hello,1> <you,1> <bye,1> <you,1>

a given *key* is allocated to a given *server*

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)

reduce(word, partCount):
  int result = 0
  for pc in partCount:
     result += pc
  emit(result)
```

map(file1, “hello me, goodbye me”)→
<hello,1> <me,1> <goodbye,1> <me,1>

map(file2, “hello you, bye you”)→
<hello,1> <you,1> <bye,1> <you,1>

a given *key* is allocated to a given *server*

reduce(hello,<1,1>) → 2
...

Example : Word Count

```
map(filename, content):
  for each w in content:
     emitInt(w, 1)

reduce(word, partCount):
  int result = 0
  for pc in partCount:
     result += pc
  emit(result)
```

map(file1, “hello me, goodbye me”)→
<hello,1> <me,1> <goodbye,1> <me,1>

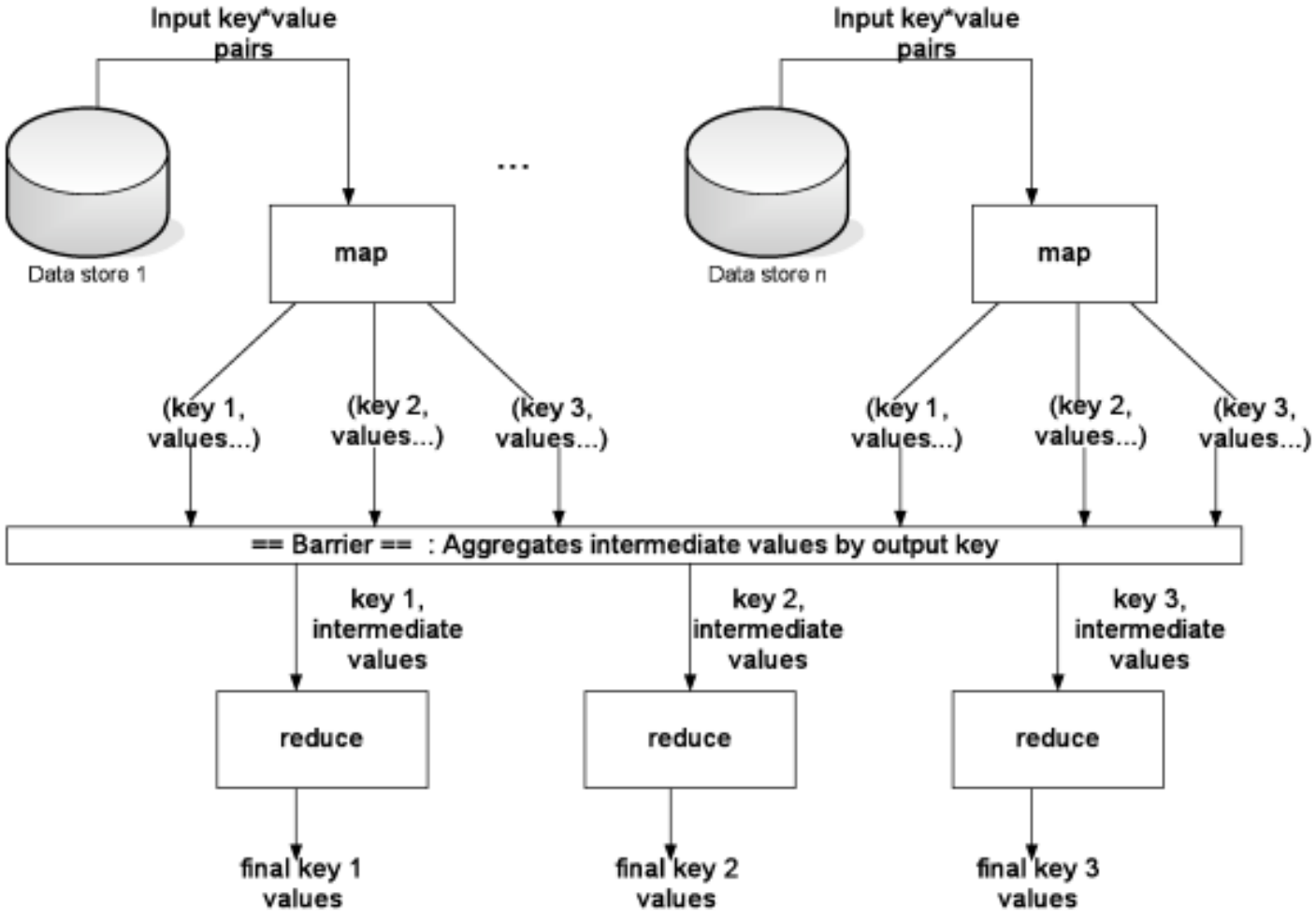
map(file2, “hello you, bye you”)→
<hello,1> <you,1> <bye,1> <you,1>

a given *key* is allocated to a given *server*

reduce(hello,<1,1>) → 2
...

<hello,2> <me,2> <you,2> <goodbye,1> <bye,1>

MapReduce Implemented



Avancement

A 30-sec conclusion

A new industrial era :

- computer science does really matter
- be fluent in at (least one) cloud tech. or die