First two classes:

Storage is the root of performance for big data systems

We increasingly need new big data systems

Designing systems is super complex (several years and moving targets)

We need to automate design as much as possible
what should you be doing?

READING
Get familiar with the very basics of traditional database architectures:

Get familiar with very basics of modern database architectures:

Get familiar with the very basics of modern large scale systems:

The Periodic Table of Data Structures.
Stratos Idreos, Kostas Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas Maas, Wilson Qin, Abdul Wasay, Yiyou Sun. IEEE Data Engineering Bull. Sep, 2018
Preparing for presentations and reviews

review and slides should answer:

what is the problem
why is it important
why is it hard
why existing solutions do not work
what is the core intuition for the solution
solution step by step
does the paper prove its claims
exact setup of analysis/experiments
are there any gaps in the logic/proof
possible next steps

* follow a few citations to gain more background

Judge lectures based on our guidelines:
how to prepare slides

no bullets   2 colors   big text   images   animation for examples
how to prepare slides

no bullets  2 colors  big text  images  animation for examples

story

one message per slide  connection from slide to slide
Today:

Self-designing systems in more detail
Goal: concept, (3) steps needed

Understanding the first step: design primitives (data structures)
More details on data structures and key-values stores
4 very high-level ways to present the same thing

How can we design complex systems automatically?
How many and which designs are possible?
How many and which designs are possible?

Can we compute performance w/o coding?
3 ALL GOOD IDEAS IN THE 60s?

**EVERY DESIGN:**

1. **A SET OF CONCEPTS**
2. **EXISTING OR NEW CONCEPTS**

**INDEX**

- scan, random access
- binary search

**DATA**

- metadata, model, function, filters
- physical layout, e.g., partitioning
EVERY DESIGN: 1 A SET OF CONCEPTS 2 EXISTING OR NEW CONCEPTS 3 ALL GOOD IDEAS IN THE 60s?
(almost) All designs are a combination/tuning of existing concepts.

Every design:
1. A set of concepts
2. Existing or new concepts
3. All good ideas in the 60s?
Nikos Kazantzakis, philosopher

I am free of action for nothing is for nothing.

The holy form of theory is ultimate fear.

I hope for nothing.
action is the most holy form of ultimate theory

I hope for nothing
I fear nothing
I am free

Nikos Kazantzakis, philosopher
action is the ultimate theory

I hope for nothing
I fear nothing
I am free

NEW
I hope for nothing
I fear nothing
I am free

Nikos Kazantzakis, philosopher
CEREAL MILK PANNA COTTA
non obvious valid combinations

milk + cream + sugar + vanilla/lemon

Christina Tosi
Best researchers: kids, young students, adults that stay kids

CEREAL MILK PANNA COTTA
non obvious valid combinations

milk + cream + sugar + vanilla/lemon
NP hard problem:
2 PhD parents trying to get a toddler to wear gloves
NP hard problem:
2 PhD parents trying to get a toddler to wear gloves
NP hard problem:
2 PhD parents trying to get a toddler to wear gloves
what is creativity?

Plato

Leibniz
fundamental building blocks
properties when combined
fundamental building blocks when combined
Three steps required

DESIGN SPACE

COST SYNTHESIS

WHAT-IF
1. DESIGN SPACE

data layout of data structures
algorithm design
systems: interactions of components

$>10^{100}$
1. DESIGN SPACE
   data layout of data structures
   algorithm design
   systems: interactions of components

2. NAVIGATE SEARCH SPACE
   cost synthesis: computation and data movement
   learned cost models in memory/parallelism
   design continuums to shrink space
Step 1:  
First Principles of Design to Construct Design Space
trial and error
FIRST PRINCIPLE: design concept that does not break further
FIRST PRINCIPLE: design concept that does not break further

KNOWN DESIGNS

OPEN QUESTIONS
To think about design principles/design space we need to first fully understand an area in extreme detail.
To think about design principles/design space we need to first fully understand an area in extreme detail

next: 2 data structures that drive 90% of modern kv-stores
B-tree & LSM-tree
Page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages
page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages
info to navigate lower level value-pointer

page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages

1,2,3… 12,15,17 20,… 
<12 >=12

12,20 35,… 50,…
page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages

info to navigate lower level value-pointer

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages

N elements, P pages
can index 8K pages
of the next level
The diagram illustrates a binary search tree with the following key components:

- **Root**: The top node with the key 35,50.
- **Internal Nodes**: Nodes with keys 12,20, 35,..., 50,...
- **Leaves**: Nodes with keys 1,2,3..., 12,15,17, 20,..., ...

- **Fanout**: The number of children a node has.
- **Height**: The height of the tree, which is $\log_{\text{fanout}} P$.

The tree's structure is organized based on the fanout, with each node branching out into at most two children. The height of the tree is determined by the fanout, illustrating the logarithmic relationship between the fanout and the tree's height.
The diagram illustrates a tree structure with various nodes and leaves. The height of the tree, denoted as $\log_{\text{fanout}} P$, is shown as the vertical distance from the root to the leaves. The root node is labeled with a range of values, possibly representing a key or shared data range. The internal nodes are connected with arrows indicating the direction of data flow or access operations. The leaves are also labeled with ranges, suggesting they represent the endpoints or final data access points.

Key terms and concepts include:
- **Random accesses**: Indicated by an arrow pointing to the tree, suggesting the dynamic nature of data access in the context of the tree structure.
- **Fanout**: A term likely referring to the branching factor of the tree, indicating how many nodes or branches can be accessed from a single node.
- **Height**: The vertical distance from the root to the leaves, a critical factor in determining the efficiency of data searches in such tree structures.
- **Leaves**: The terminal nodes of the tree, marked with ranges of values, suggesting they are the final data points or access endpoints.
- **Internal nodes**: The non-terminal nodes that connect the root to the leaves, essential for navigating through the tree structure.

The diagram is part of a presentation or educational material from DASlab @ Harvard SEAS, suggesting a focus on data structures and algorithms, possibly within the domain of computer science or information technology.
leaves

get 15
get 15-55
leaves

how do we search the leaves?

a) sorted and b) unsorted leaves
NoSQL Key-value Stores

- RocksDB
- Google BigTable
- mongoDB
- SQLite
- LinkedIn
- Amazon DynamoDB
- cassandra
- Apache HBase

Applications:
- Machine learning
- Social media
- Smart homes
- Web browsers
- Phones
- Web-based apps
- Security
- Health devices
- Graphs
- Analytics
insert (key-value)
Level 1
insert (key-value)

buffer

Level 1

MEMORY

DISK
insert (key-value)
insert (key-value)

buffer

Level 1

Level 2

Level 3

...

Level N
insert (key-value)
insert (key-value)

buffer

Level 1

Level 2

Level 3

... 

Level N

MEMORY DISK

pages

SSTables

tiered

leveled

sorted
get (key)

buffer

[1,0,0,1,1,1] hash fun.
bloom filters

[min-max] /page
fence pointers

Level 1

Level 2

Level 3

... ... ...

Level N

MEMORY

DISK

pages

SSTables

 leveled tiered sorted
get (key)

buffer

Level 1

Level 2

Level 3

Level N

[1,0,0,1,1,1] hash fun.
bloom filters

[min-max] /page
fence pointers

MEMORY

DISK

pages
SSTables

leveled tiered

sorted

hash fun.
fence pointers

get (key)
get (key)

buffer

Level 1

Level 2

Level 3

... Level N

bloom filters

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

MEMORY

DISK

pages

SSTables

leveled tiered sorted

hash fun.

fence pointers

get (key)
get (key)

buffer

Level 1

Level 2

Level 3

... Level N

SSTables

pages

MEMORY

DISK

tiered

leveled

sorted

[min-max] /page

bloom filters

hash fun.

fence pointers

[1,0,0,1,1,1]
get (key)

buffer

Level 1

Level 2

Level 3

Level N

MEMORY

DISK

pages

SSTables

tiered

leveled

sorted

bloom filters

fence pointers

[min-max] /page

[1,0,0,1,1,1] hash fun.
bloom filters  |  fence pointers  |
[1,0,0,1,1,1] hash fun.  |  [min-max] /page  |

Level 1
Level 2
Level 3
Level N

MEMORY
DISK
SSTables
pages

[1,0,0,1,1,1] hash fun.
[min-max] /page

hash fun.
fence pointers
buffers

Level 1
Level 2
Level 3
Level N

sorted
levelled

tiered
What do we want to achieve: what if design example
workload, h/w, layout, design
without coding or accessing the h/w
without coding or accessing the h/w layout design

workload

h/w

algorithms

performance

what-if design.
What if I add bloom filters to my B-tree? accessing the h/w
What if I add bloom filters to my B-tree?

What if I add feature X that brings 60% more writes?
What if I need to reduce memory by 50%?

What if I add bloom filters to my B-tree?

What if I add feature X that brings 60% more writes?
Cost in Amazon Cloud?  
What if I add bloom filters to my B-tree?  
Which workload breaks my system?  
What if add feature X that brings 60% more writes?  
Should I buy new hardware X?  
What if I need to reduce memory by 50%?  

what-if design
Reading for KV background & systems project


**Modern B-Tree Techniques**
by Goetz Graefe
Foundations and Trends in Databases, 2011
Sections: 1,2,3,5