https://hao-ai-lab.github.io/dsc204a-w24/

DSC 204A: Scalable Data Systems
Winter 2024

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

Recap

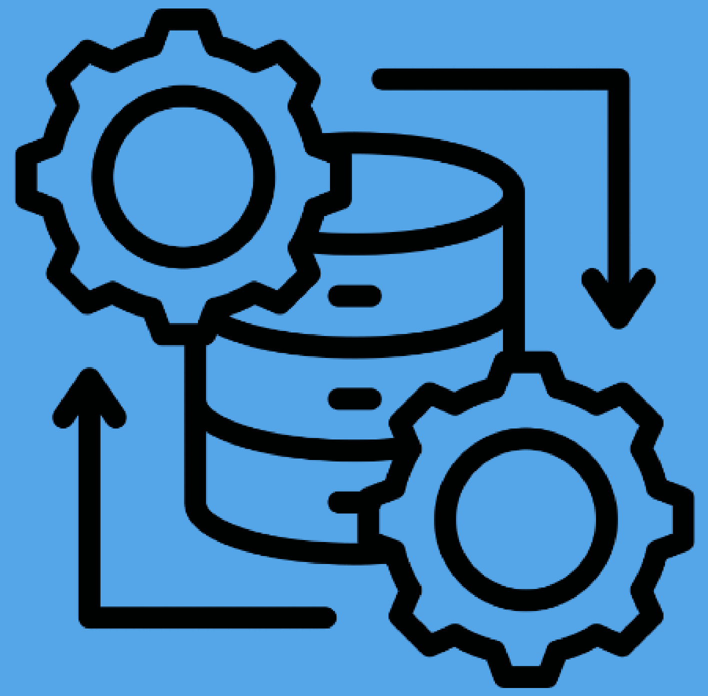
OLAP

- Analytical
- Show queries
- Denormalised
- Historical Data

BUSINESS DATA WAREHOUSE

OLTP

- Transactional
- Fast Processing
- Normalised
- Current Data



BUSINESS PROCESS

How should OLTP database be improved to accommodate OLAP use?

Today's topic: Column-oriented storage and schemas

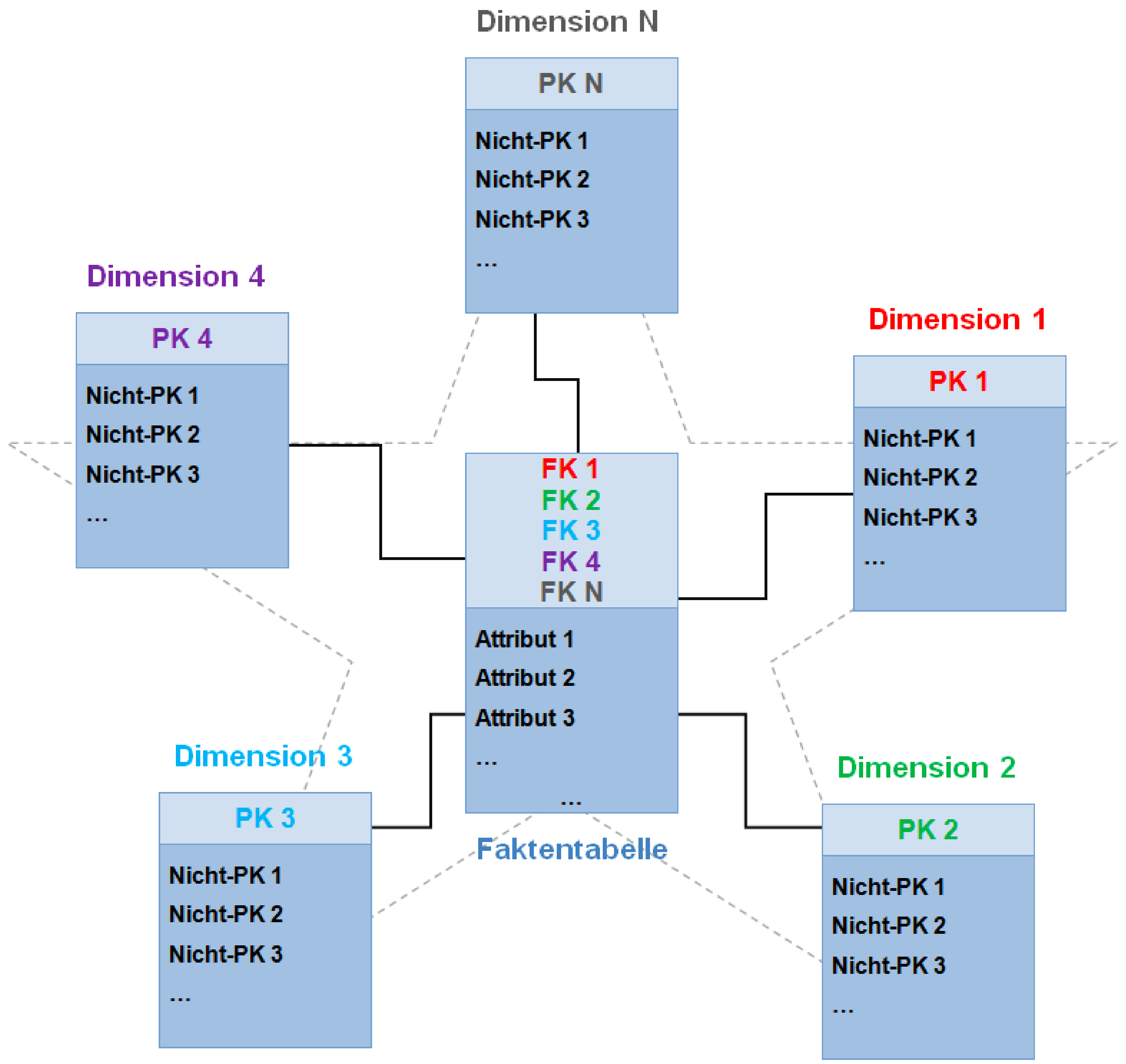
- OLTP v.s. OLAP
- Data warehousing
- Schemas for Analytics
- Column-oriented storage

Data analytic queries

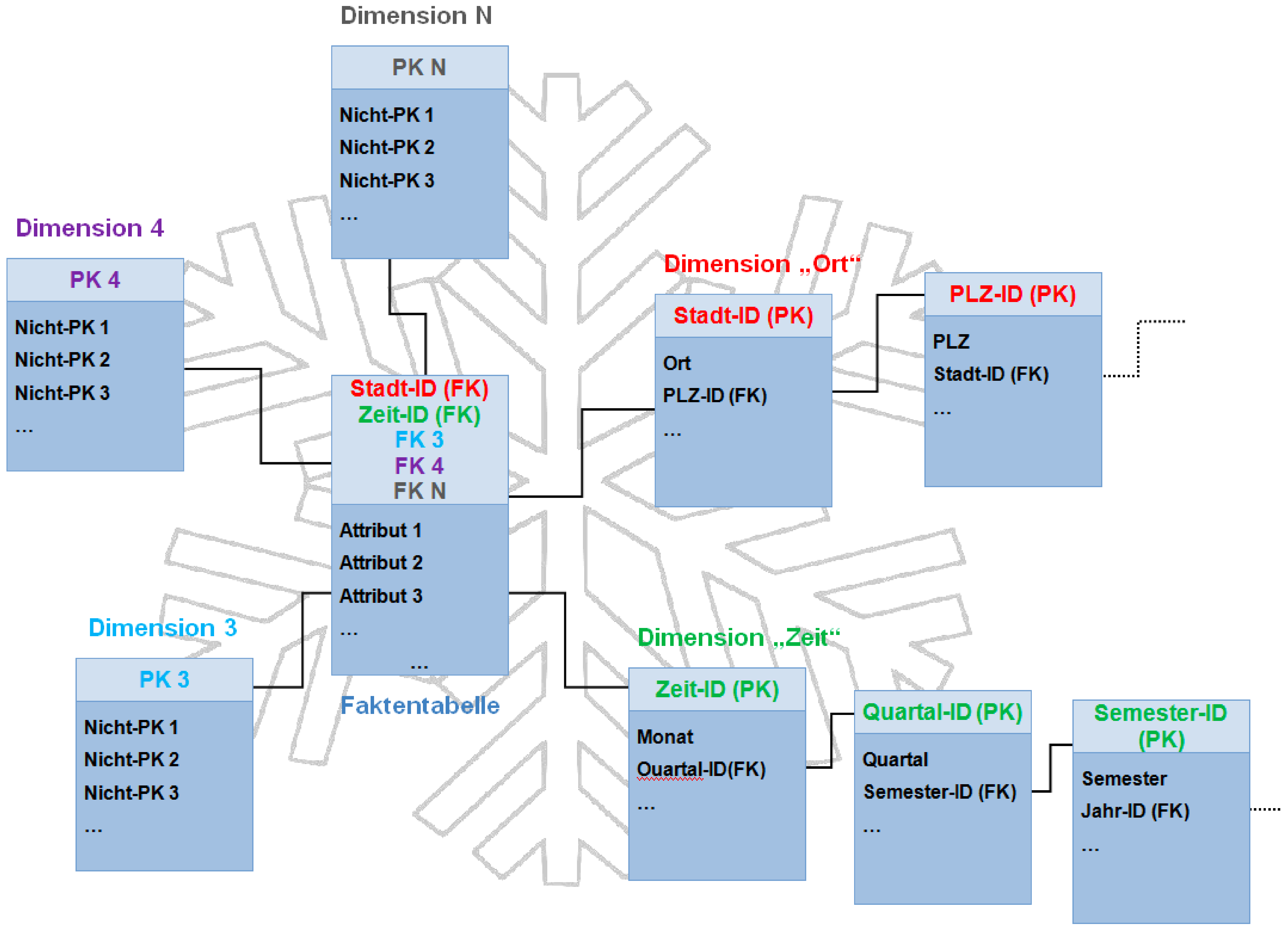
- What was the total revenue of each of our stores in Jan?
- How many more bananas that usual did we sell during our latest data?
- Which brand of baby food is most often purchased together with brand X diapers?

Popular Schemas in Database (DSC 102)

- Relation (SQL)
- Document (NoSQL)
- Graph (GraphQL)
- Network
- Hierarchy
- **Stars**
- **Snowflake**



Star



Snowflake

Star schema

- Fact table in the middle
  - A collection of events
  - e.g., click events, page views, retail sales
  - Two types of columns
    - Attributes
    - References to dimension tables.
- Fact table: event meta data
- Dimensions: who, what, where, when, how, and why of the event.

dim_product table

product_sk	sku	description	brand	category
30	OK4012	Bananas	Freshmax	Fresh fruit
31	KA9511	Fish food	Aquatech	Pet supplies
32	AB1234	Croissant	Dealicious	Bakery

dim_store table

store_sk	state	city
1	WA	Seattle
2	CA	San Francisco
3	CA	Palo Alto

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	31	3	NULL	NULL	1	2.49	2.49
140102	69	5	19	NULL	3	14.99	9.99
140102	74	3	23	191	1	4.49	3.89
140102	33	8	NULL	235	4	0.99	0.99

dim_date table

date_key	year	month	day	weekday	is_holiday
140101	2014	jan	1	wed	yes
140102	2014	jan	2	thu	no
140103	2014	jan	3	fri	no

dim_customer table

customer_sk	name	date_of_birth
190	Alice	1979-03-29
191	Bob	1961-09-02
192	Cecil	1991-12-13

dim_promotion table

promotion_sk	name	ad_type	coupon_type
18	New Year sale	Poster	NULL
19	Aquarium deal	Direct mail	Leaflet
20	Coffee & cake bundle	In-store sign	NULL

Example: dim_date table

- Speed up the analysis.
- Easier development.

dim_date table

date_key	year	month	day	weekday	is_holiday
140101	2014	jan	1	wed	yes
140102	2014	jan	2	thu	no
140103	2014	jan	3	fri	no

Today's topic: Column-oriented storage

- OLTP v.s. OLAP
- Data warehousing
- Schemas for Analytics
- Column-oriented storage

Data scale

- Fact tables
  - Hundreds of columns
  - Trillions of rows
  - Petabytes of data
- Dimension tables
  - Million of rows.
  - Can be wide. But less common.

How many columns do we need?

- What was the total revenue of each of our stores in Jan?
- How many more bananas that usual did we sell during our latest data?
- Which brand of baby food is most often purchased together with brand X diapers?

dim_product table

product_sk	sku	description	brand	category
30	OK4012	Bananas	Freshmax	Fresh fruit
31	KA9511	Fish food	Aquatech	Pet supplies
32	AB1234	Croissant	Dealicious	Bakery

dim_store table

store_sk	state	city
1	WA	Seattle
2	CA	San Francisco
3	CA	Palo Alto

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	31	3	NULL	NULL	1	2.49	2.49
140102	69	5	19	NULL	3	14.99	9.99
140102	74	3	23	191	1	4.49	3.89
140102	33	8	NULL	235	4	0.99	0.99

dim_date table

date_key	year	month	day	weekday	is_holiday
140101	2014	jan	1	wed	yes
140102	2014	jan	2	thu	no
140103	2014	jan	3	fri	no

dim_customer table

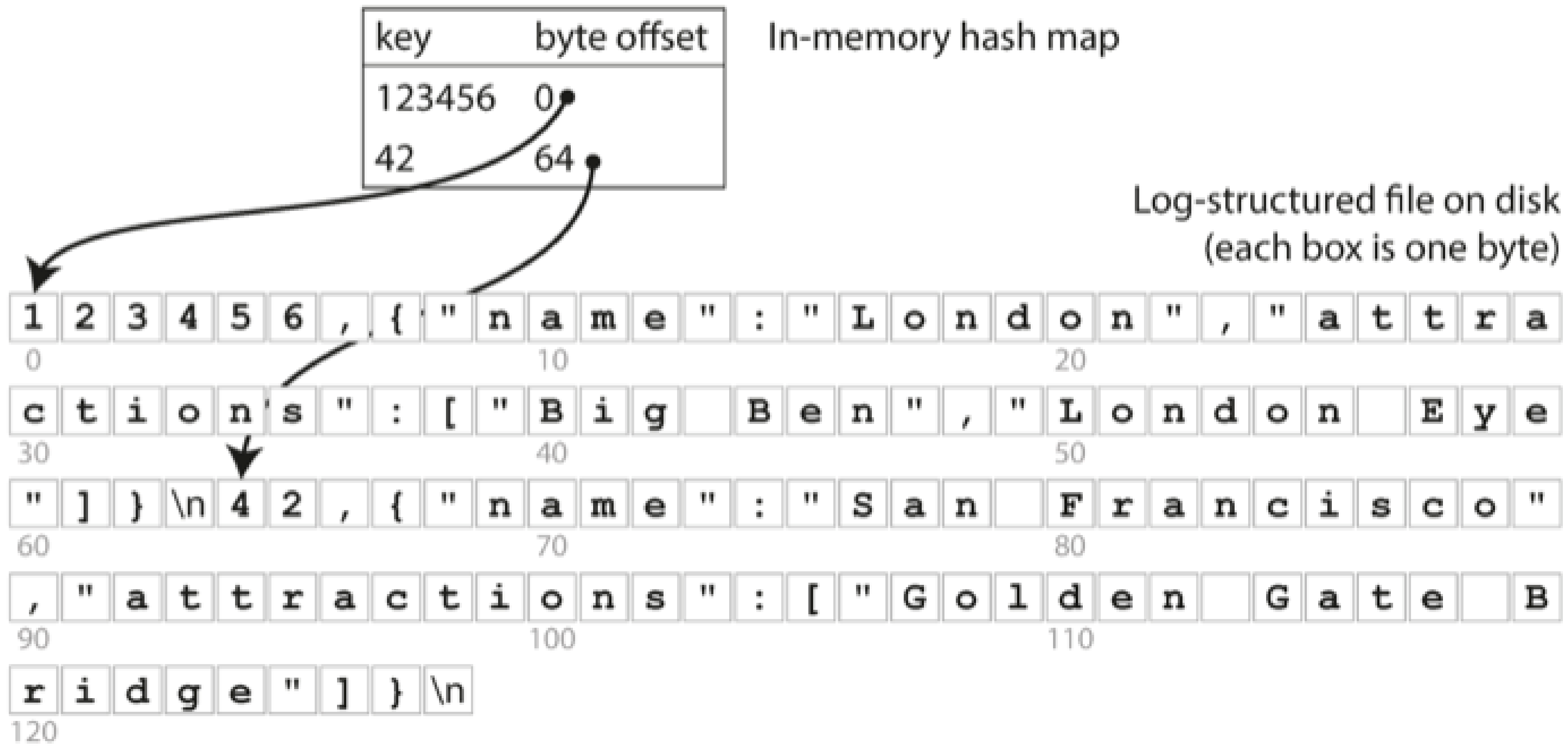
customer_sk	name	date_of_birth
190	Alice	1979-03-29
191	Bob	1961-09-02
192	Cecil	1991-12-13

dim_promotion table

promotion_sk	name	ad_type	coupon_type
18	New Year sale	Poster	NULL
19	Aquarium deal	Direct mail	Leaflet
20	Coffee & cake bundle	In-store sign	NULL

How does a Row-oriented storage work?

- Storage:
  - All the values from one row of a table are stored next to each other.
- What was the total revenue in January?
  - Load indexes into the memory
  - Find all the records in January.
  - Load all of these rows (100+ attributes) from disk into memory
  - Parse
  - Filter



Implementations for column-oriented

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date_key file contents:	140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents:	69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents:	4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents:	NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents:	NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents:	1, 3, 1, 5, 1, 3, 1, 1
net_price file contents:	13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents:	13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

How does a column-oriented storage work?

- Storage:
  - Store all the values from each column together instead.
- What was the total revenue in January?
  - Load indexes into the memory
  - Find all the records in January.
  - Load all of these rows (~~100+ attributes~~ -> 1 row) from disk into memory
    - 100 times improvement
  - Parse
  - Filter

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date_key file contents:	140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents:	69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents:	4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents:	NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents:	NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents:	1, 3, 1, 5, 1, 3, 1, 1
net_price file contents:	13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents:	13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Column compression

- Data in the same column are more repetitive.
- Save storage space
- Improve I/O bandwidth usage

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date_key file contents:	140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents:	69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents:	4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents:	NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents:	NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents:	1, 3, 1, 5, 1, 3, 1, 1
net_price file contents:	13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents:	13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Bitmap encoding

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

- Many transactions
- A small amount of distinct values
- Transform them into bitmaps
  - Bitmap => One unique value
  - Bit => Occurrence & Order

Bitmap encoding

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Run-length encoding:

product_sk = 29:	9, 1	(9 zeros, 1 one, rest zeros)
product_sk = 30:	10, 2	(10 zeros, 2 ones, rest zeros)
product_sk = 31:	5, 4, 3, 3	(5 zeros, 4 ones, 3 zeros, 3 ones, rest zeros)
product_sk = 68:	15, 1	(15 zeros, 1 one, rest zeros)
product_sk = 69:	0, 4, 12, 2	(0 zeros, 4 ones, 12 zeros, 2 ones)
product_sk = 74:	4, 1	(4 zeros, 1 one, rest zeros)

- Many transactions
- A small amount of distinct values
- Transform them into bitmaps
  - Bitmap => One unique value
  - Bit => Occurrence & Order
- Run-length encoding for sparse bitmaps

Bitmap indexes for queries

`WHERE product_sk IN (30, 68, 69):`

Load the three bitmaps for `product_sk = 30`, `product_sk = 68`, and `product_sk = 69`, and calculate the bitwise *OR* of the three bitmaps, which can be done very efficiently.

`WHERE product_sk = 31 AND store_sk = 3:`

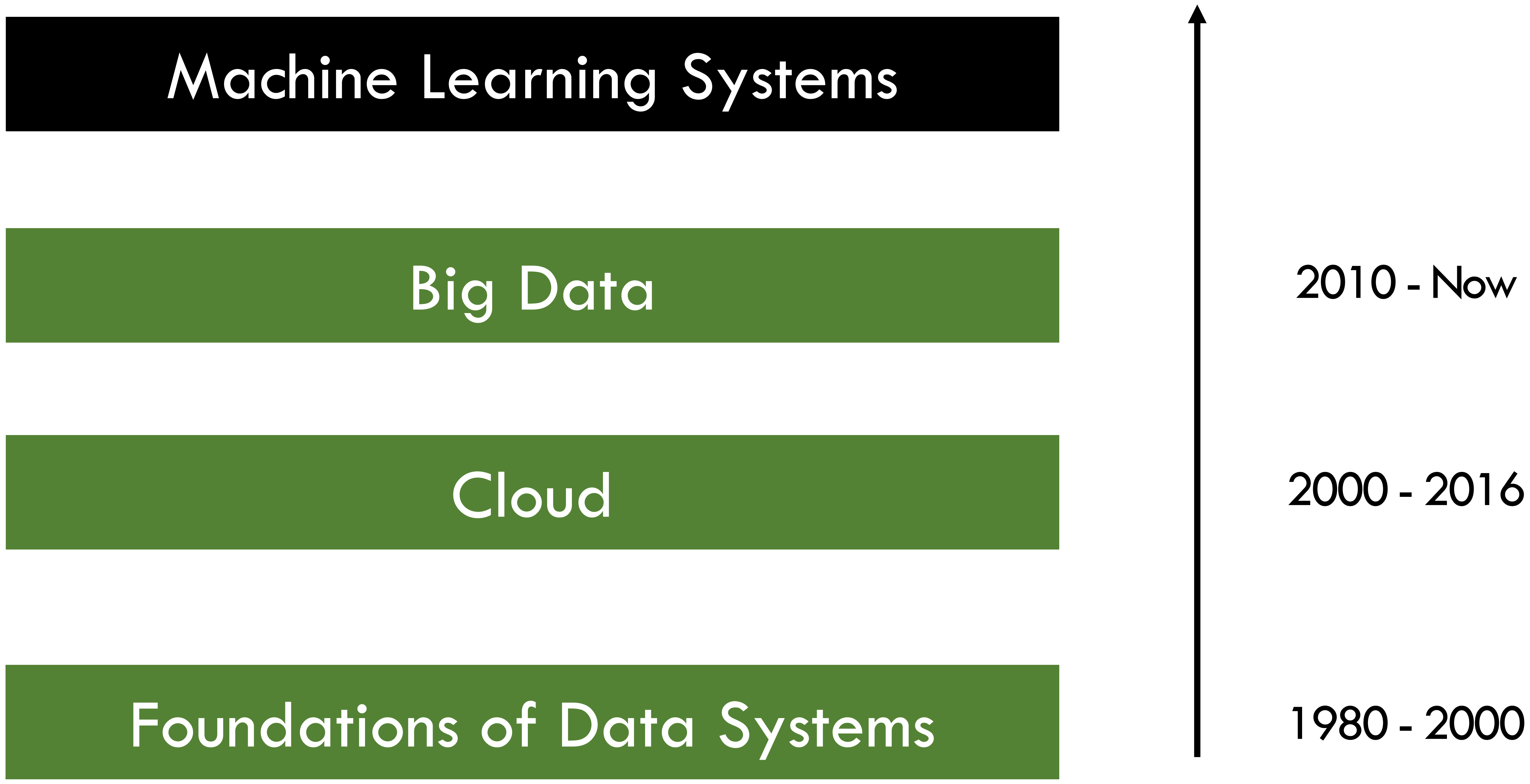
Load the bitmaps for `product_sk = 31` and `store_sk = 3`, and calculate the bitwise *AND*. This works because the columns contain the rows in the same order, so the *k*th bit in one column's bitmap corresponds to the same row as the *k*th bit in another column's bitmap.

Bitwise operations?

Summary of Important Concepts

- Hashtable indexes, SSTable, LSM, B-tree
  - Self-balanced tree
  - Bloom Filter
- General rule of thumbs:
  - LSM Tree -> in-memory database
  - B-Tree -> classic relational / on-disk database
- OLAP vs. OLTP
- Data warehouse
  - Schemas for Analytics
  - Column-oriented storage

Where We Are


Machine Learning Systems
Big Data
2010 - Now
Cloud
2000 - 2016
Foundations of Data Systems
1980 - 2000

Where We Are

Motivations, Economics, Ecosystems, Trends

Cloud

Networking		Storage		**Part3: Compute**	
~~Datacenter networking~~	Collective communication	(Distributed) File Systems / Database	Cloud storage	Distributed Computing	Big data processing

Distributed Computing and Big Data

- Parallelism Basics
- Data Replication and partitioning
- [Maybe] Consensus
- Batched Processing
- Streaming Processing
- Guest Lectures

Today’s topic: Parallelism

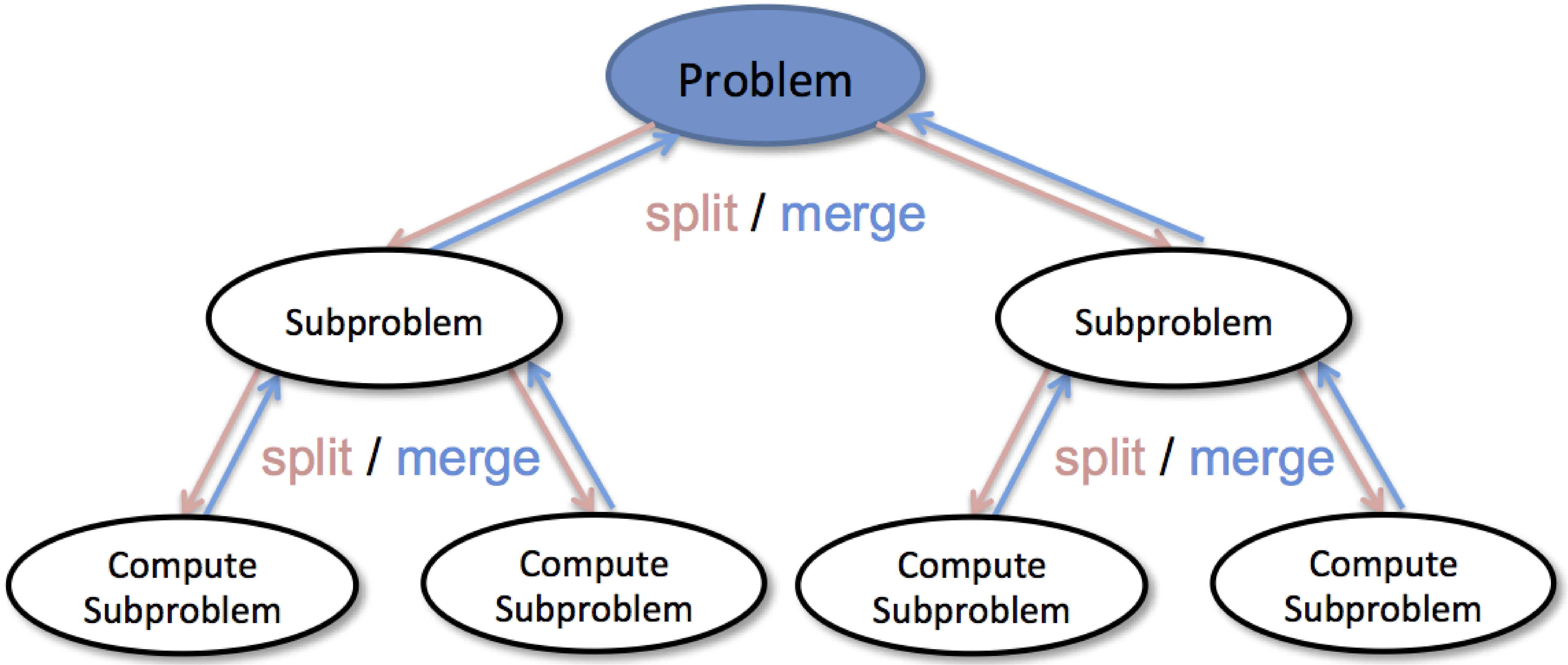
- Express data processing in abstraction
- Parallelisms
  - Task parallelism
  - Data parallelism
  - Terms: SIMD, SIMT, SPMD, MPMD

Parallel Data Processing

Central Issue: Workload takes too long for one processor!

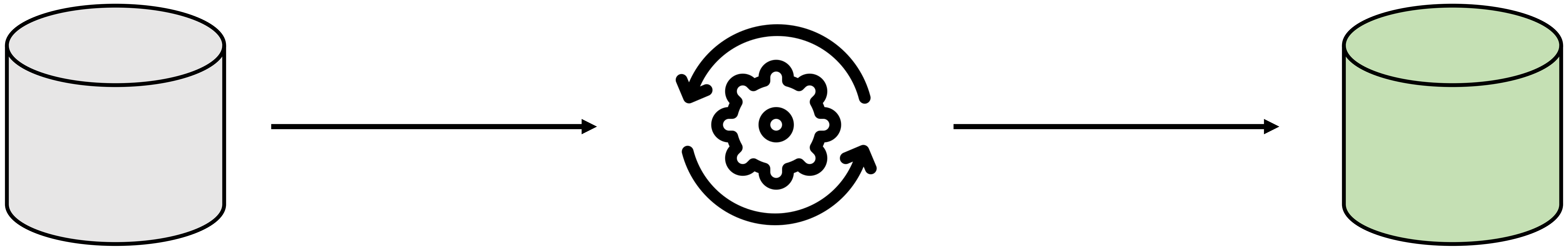
Basic Idea: Split up workload across processors and perhaps also across machines/workers (aka "Divide and Conquer")

Remind you of PA1 (hope you've enjoyed it)



https://medium.com/cracking-the-data-science-interview/divide-and-conquer-algorithms-b135681d08fc

Data Processing: Abstraction



Original data

Processing functions

- sum, mean
- Page rank
- Supervised Learning
- Clustering
- Model inference

Result data

- data
- ML models

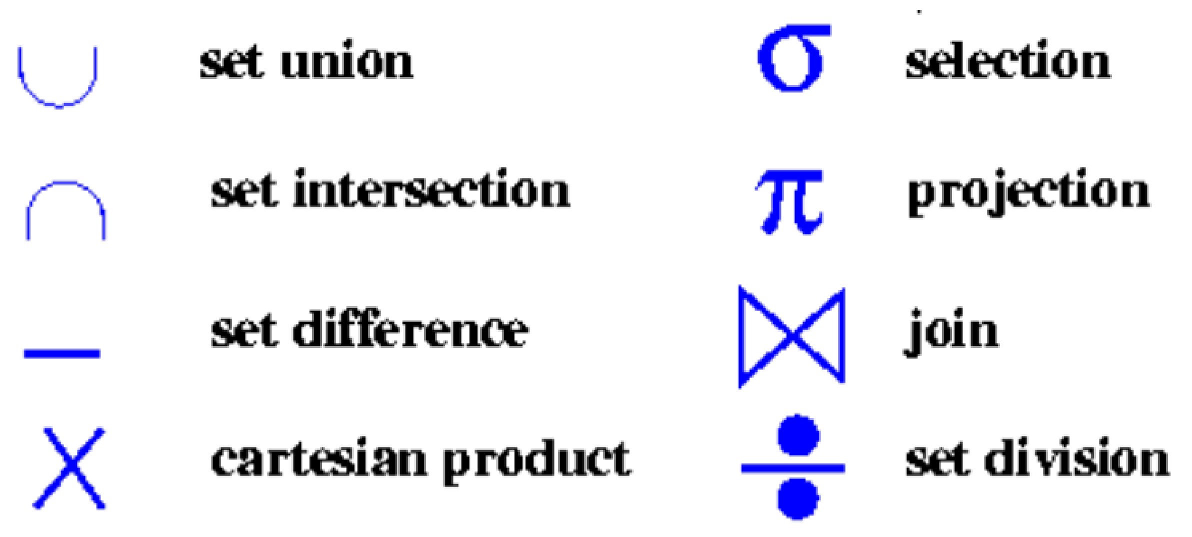
Q: How to represent various processing functions?

How to Express Arbitrarily Complex Processing Functions?

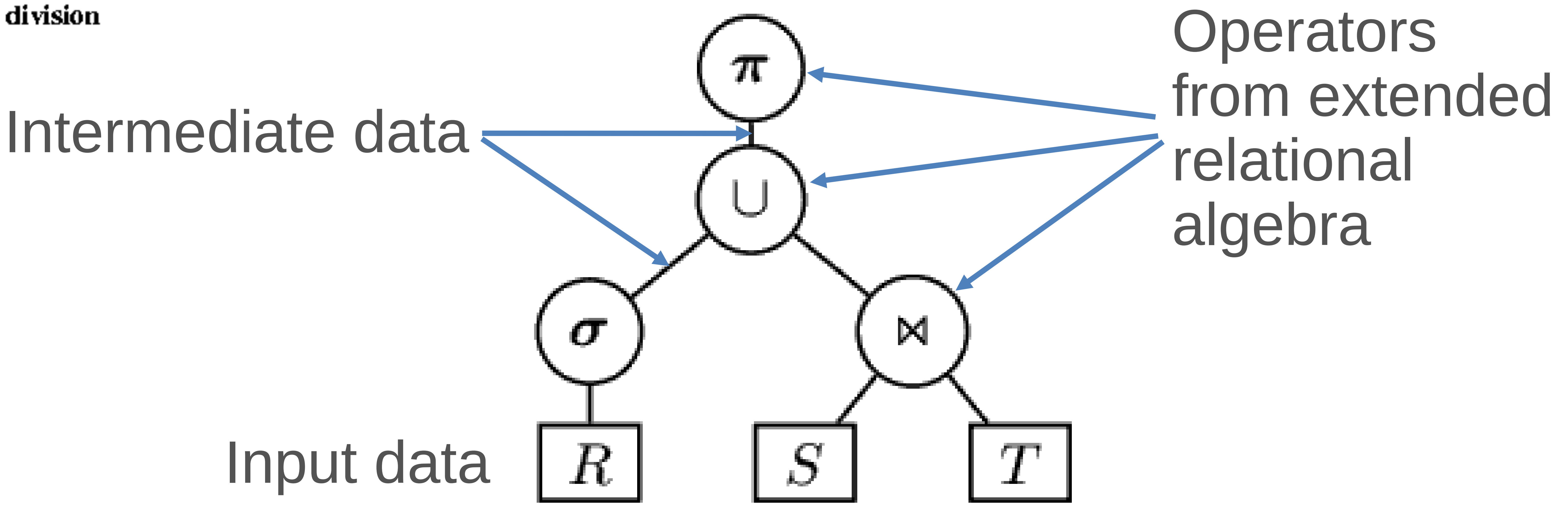
***Dataflow Graph*:** common in parallel data processing

- A ***directed*** graph representation of a program
  - **Vertices:** abstract operations from a restricted set of computational primitives:
  - **Edges:** data flowing directions (hence data dependency)
- Examples
  - Relational dataflows: RDBMS, Pandas, Modin
  - Matrix/tensor dataflows: NumPy, PyTorch, TensorFlow
- Enables us to reason about data-intensive programs at a higher level

Example: Relational Dataflow Graph



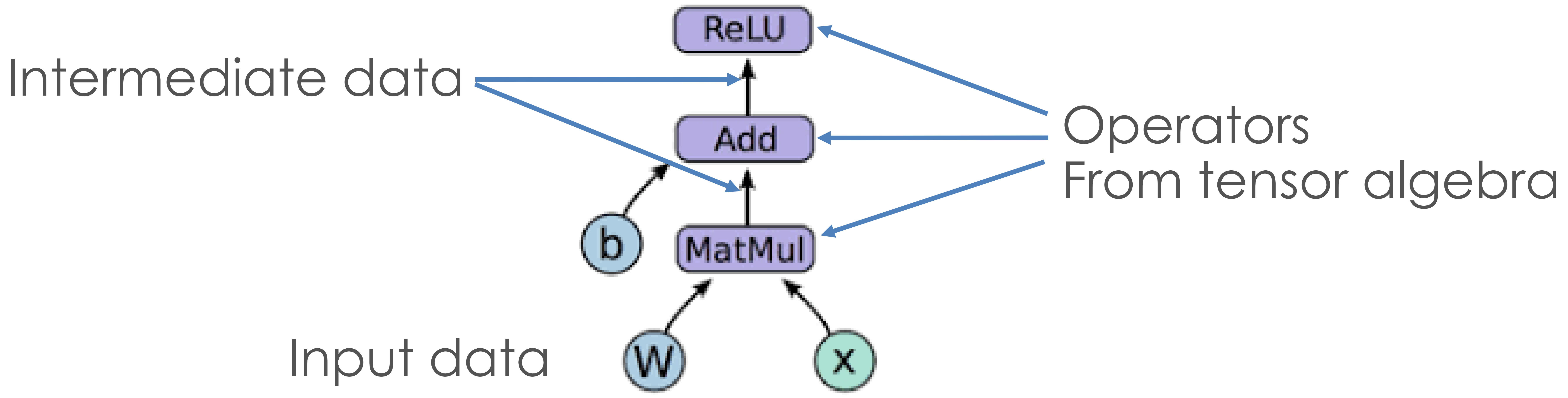
$$\pi(\sigma(R) \cup S \bowtie T)$$



Aka **Logical Query Plan** in the DB systems world

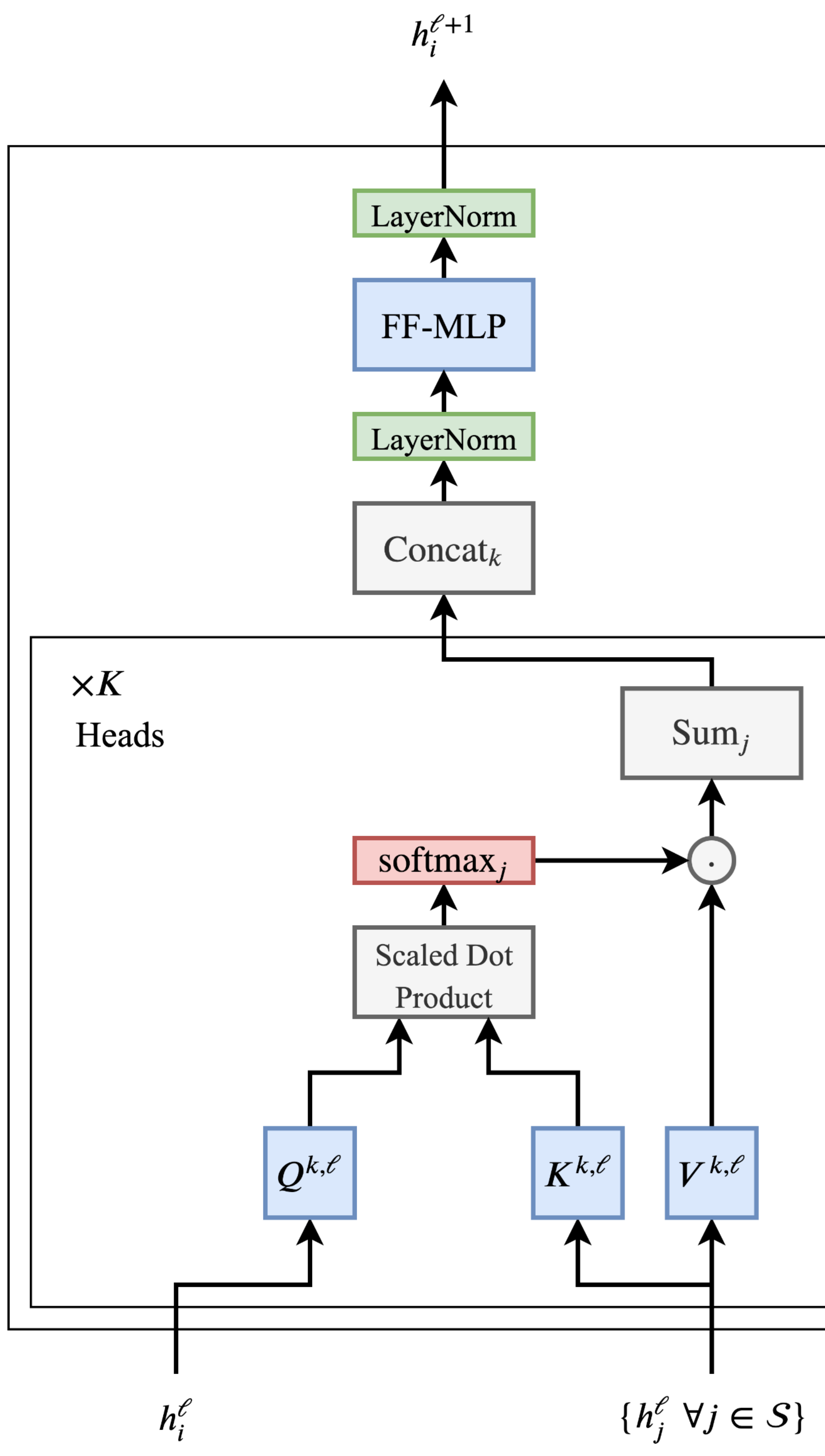
Example: Machine Learning Dataflow Graph

$$ReLU(WX + b)$$



Aka **Neural network computational graph** in ML systems

What is ChatGPT's dataflow graph Looking like?



Parallelism

Central Issue: Workload takes too long for one processor!

Basic Idea: Split up workload across processors and perhaps also across machines/workers (aka "Divide and Conquer")

Key parallelism paradigms in data systems

- **assuming there will be coordination:**

func \ data	**Shared**	**Replicated**	**Partitioned**
Replicated	N/A (rare cases)		Data parallelism
Partitioned	Task parallelism		Hybrid parallelism

Terms are confusing

- Different domains term them differently in different contexts
- Architecture/parallel computing: single-node multi-cores
  - SIMD, MIMD, SIMT
- Distributed system: multiple-node multi-cores
  - SPMD vs. MPMD
- Machine learning community
  - Data parallelism vs. Model parallelism
  - Inter-operator parallelism vs. Intra-operator parallelism

Today’s topic: Parallelism

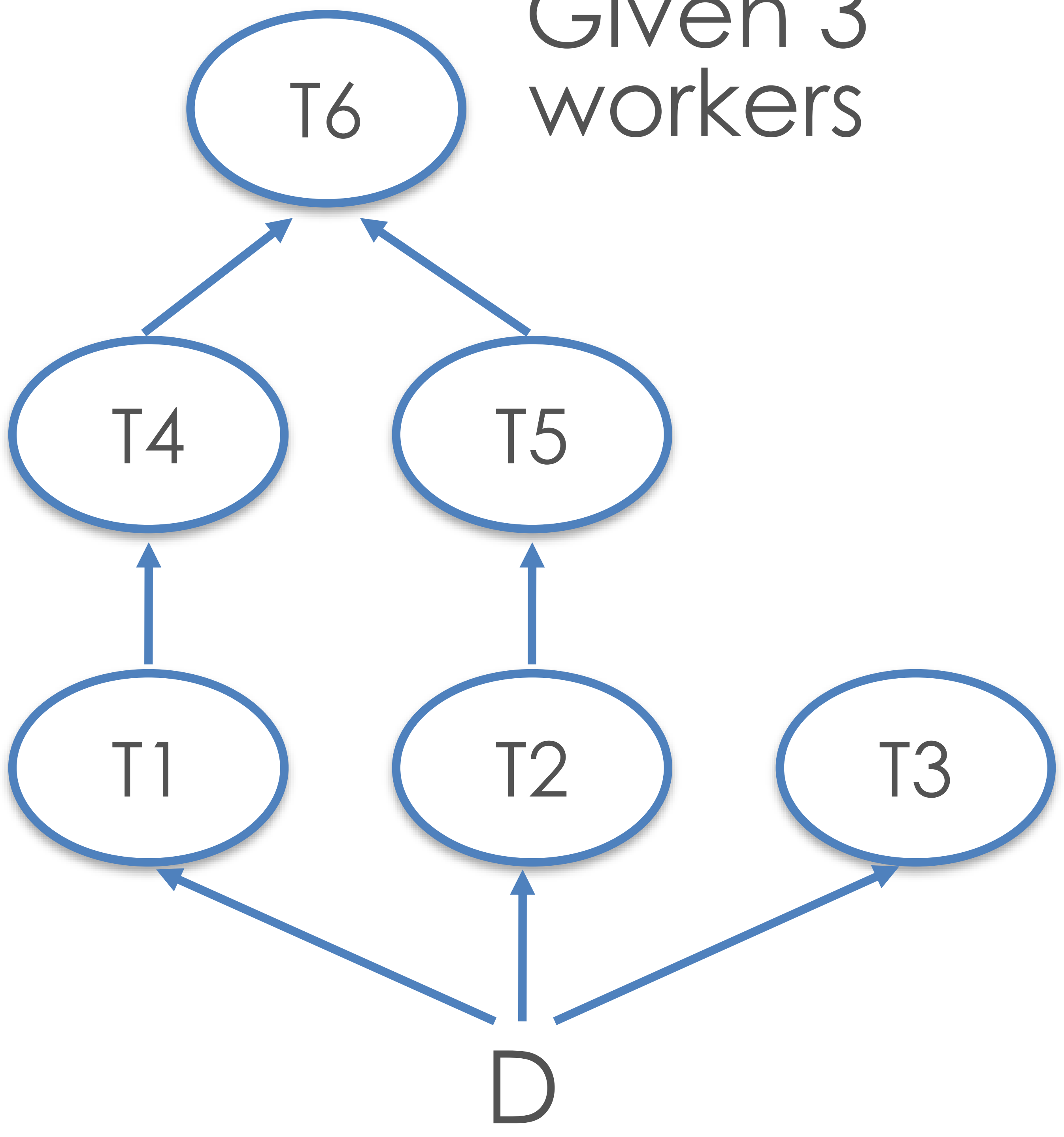
- Express data processing in abstraction
- Parallelisms
  - **Task parallelism**
  - Data parallelism
  - Terms: SIMD, SIMT, SPMD, MPMD

Task Parallelism

Basic Idea: Split up *tasks* across workers; if there is a common dataset that they read, just make copies of it (aka *replication*)

Example:

Given 3 workers



4) After T4 & T5 end, run T6 on W1; W2 is *idle*

3) After T1 ends, run T4 on W1; after T2 ends, run T5 on W2; after T3 ends, W3 is *idle*

2) Put T1 on worker 1 (W1), T2 on W2, T3 on W3; run all 3 in parallel

1) Copy whole D to all workers

Task Parallelism

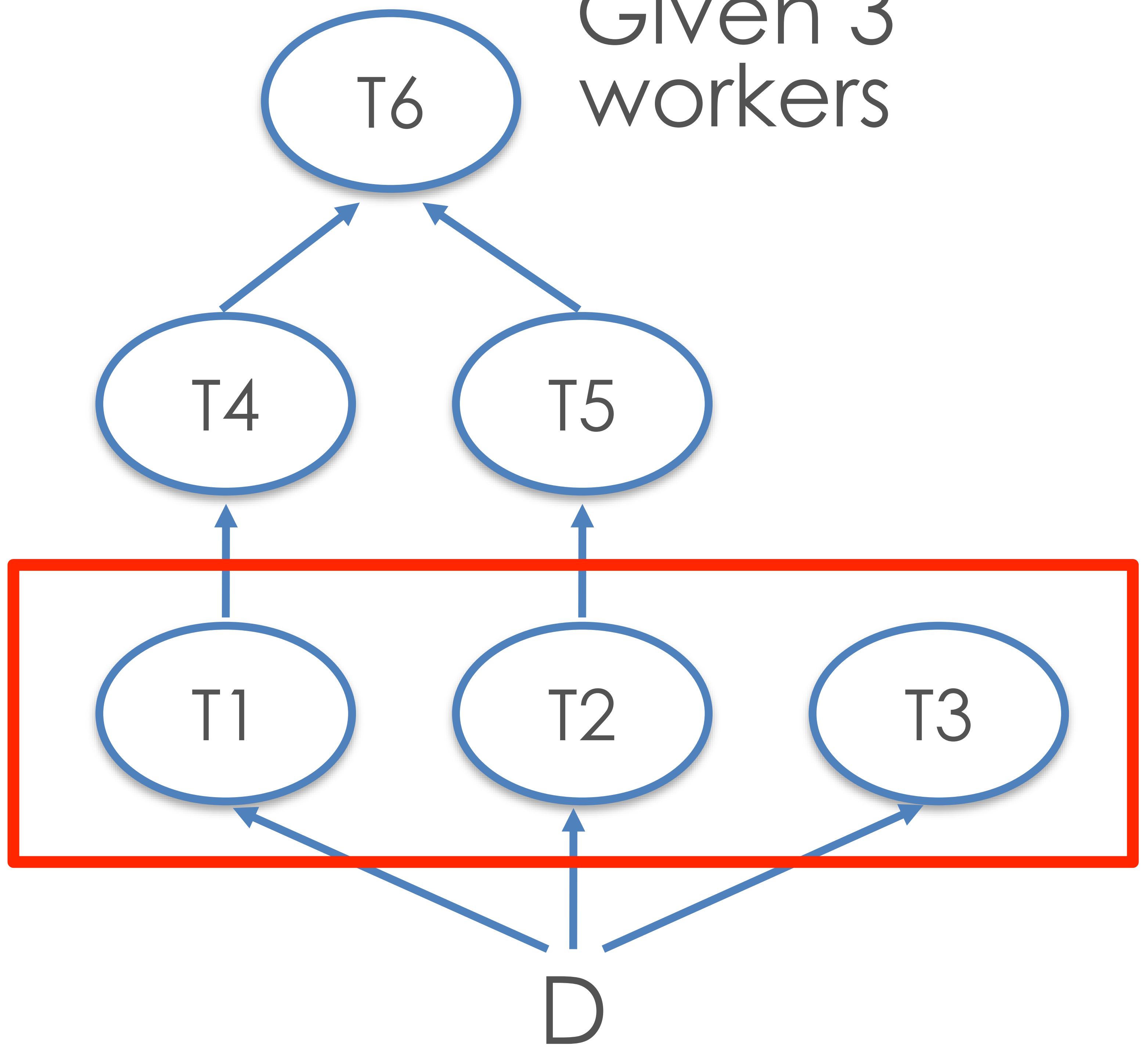
- Topological sort of tasks in task graph for scheduling
- Notion of a “worker” can be at processor/core level, not just at node/server level
  - Thread-level parallelism possible instead of process-level
  - E.g., Dask: 4 worker nodes x 4 cores = 16 workers total
- Main pros of task parallelism:
  - Simple to understand
  - Independence of workers => low software complexity
- Main cons of task parallelism:
  - Can be difficult to implement
  - Idle times possible on workers

Degree of Parallelism

- The largest amount of *concurrency* possible in the task graph, i.e., how many task can be run simultaneously

Example:

Given 3 workers



Q: *How do we quantify the runtime performance benefits of task parallelism?*

But over time, degree of parallelism keeps dropping in this example

Degree of parallelism is only 3

So, more than 3 workers is not useful for this workload!

Quantifying Benefit of Parallelism: Speedup

$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

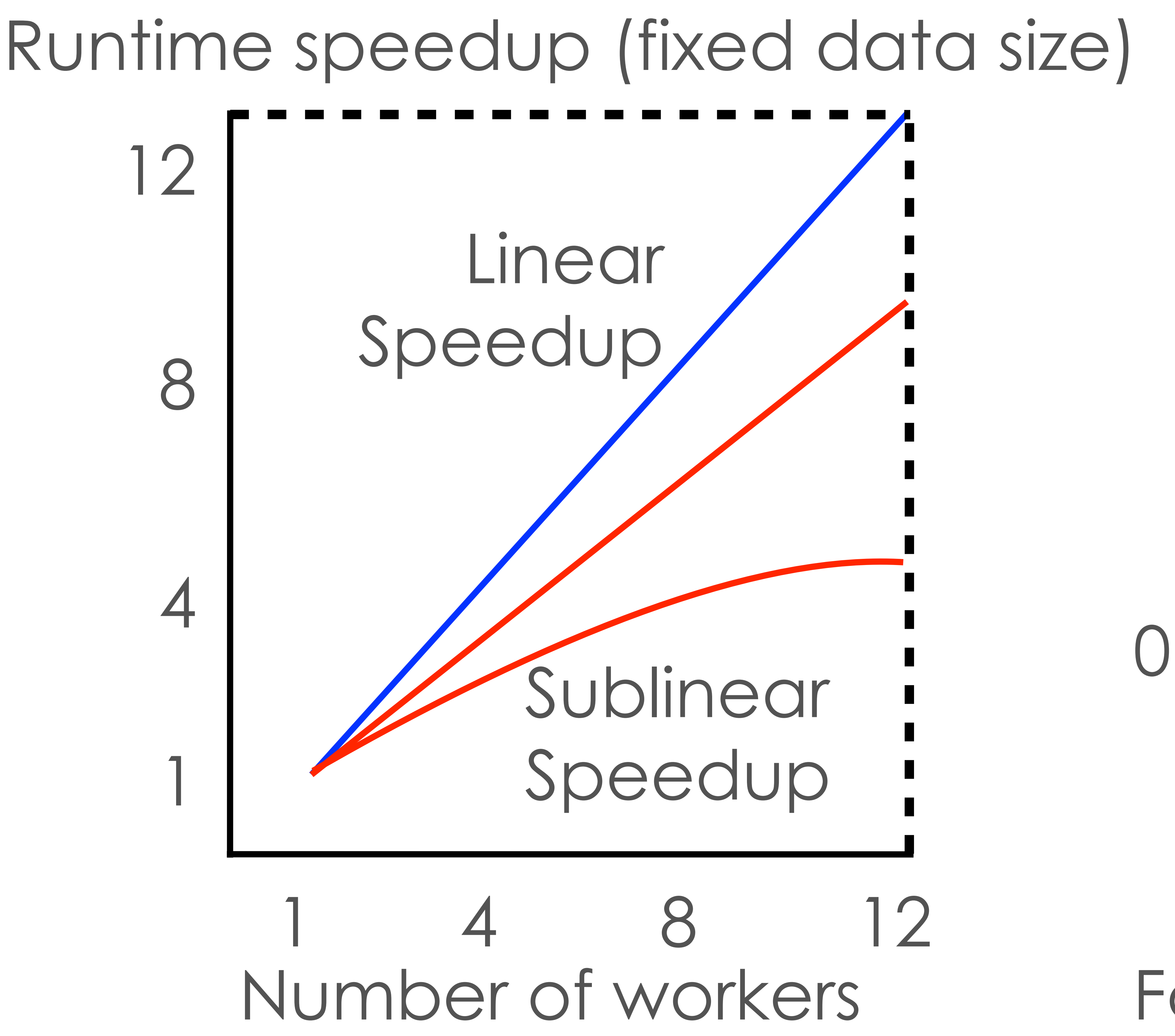
Q: *But given n workers, can we get a speedup of n?*

It depends!

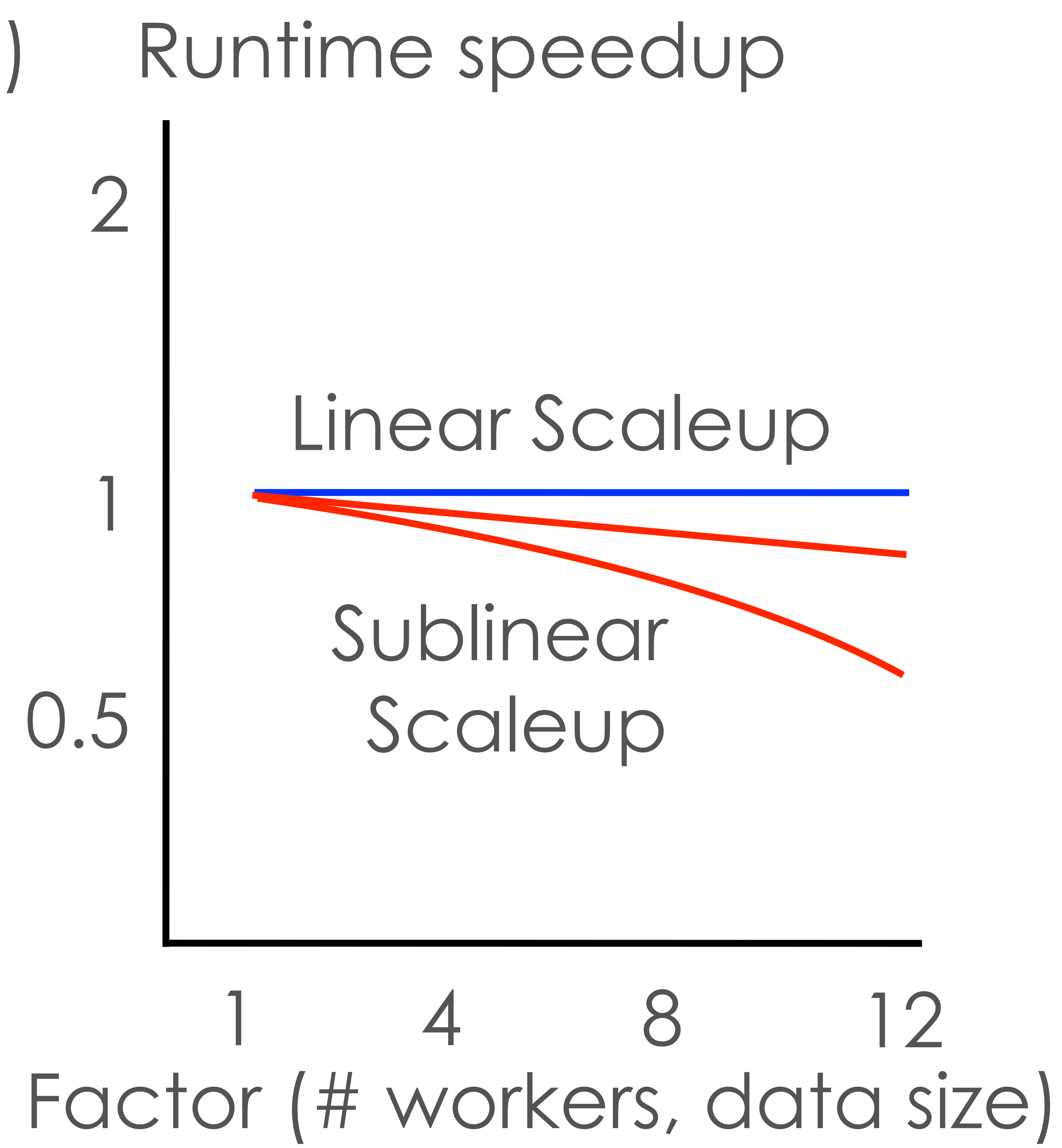
(On degree of parallelism, task dependency graph structure, intermediate data sizes, etc.)

Q: what kind of graphs can give a speedup of n?

Weak and Strong Scaling



Speedup plot / Strong scaling



Scaleup plot / Weak scaling

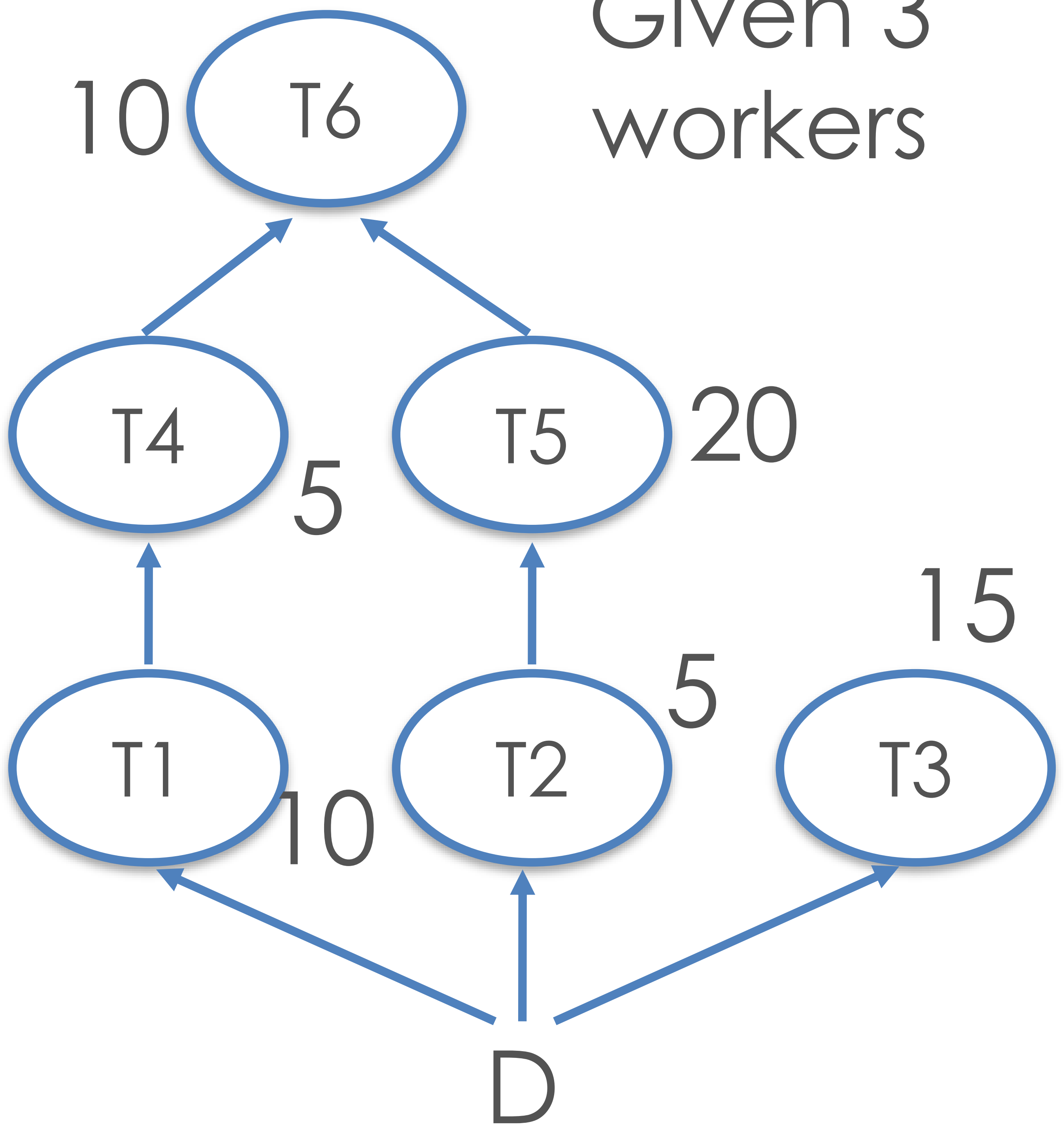
Q: *Is <u>superlinear</u> speedup/scaleup ever possible?*

Idle Times in Task Parallelism

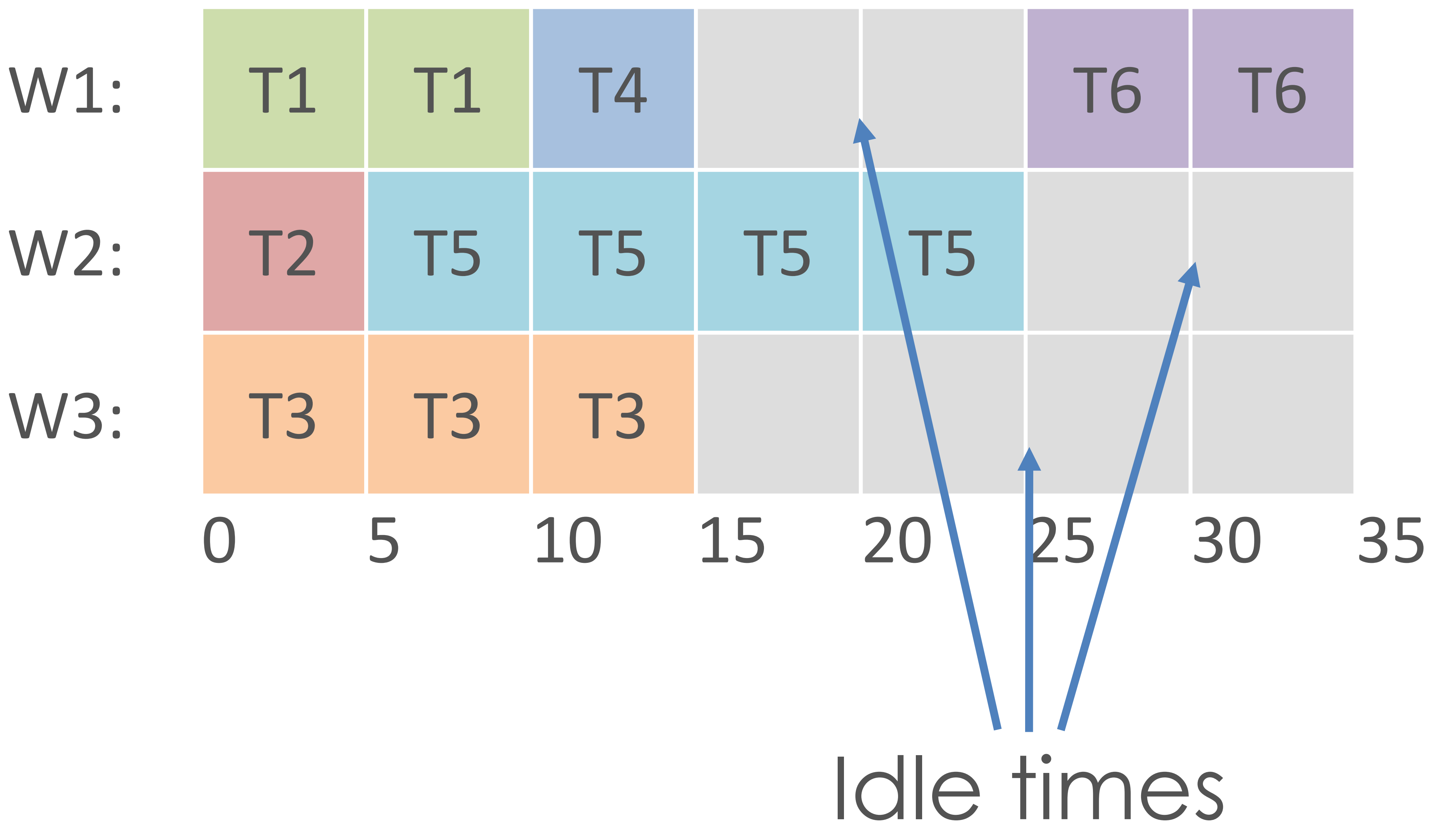
Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:

Given 3 workers



Gantt Chart visualization of schedule:

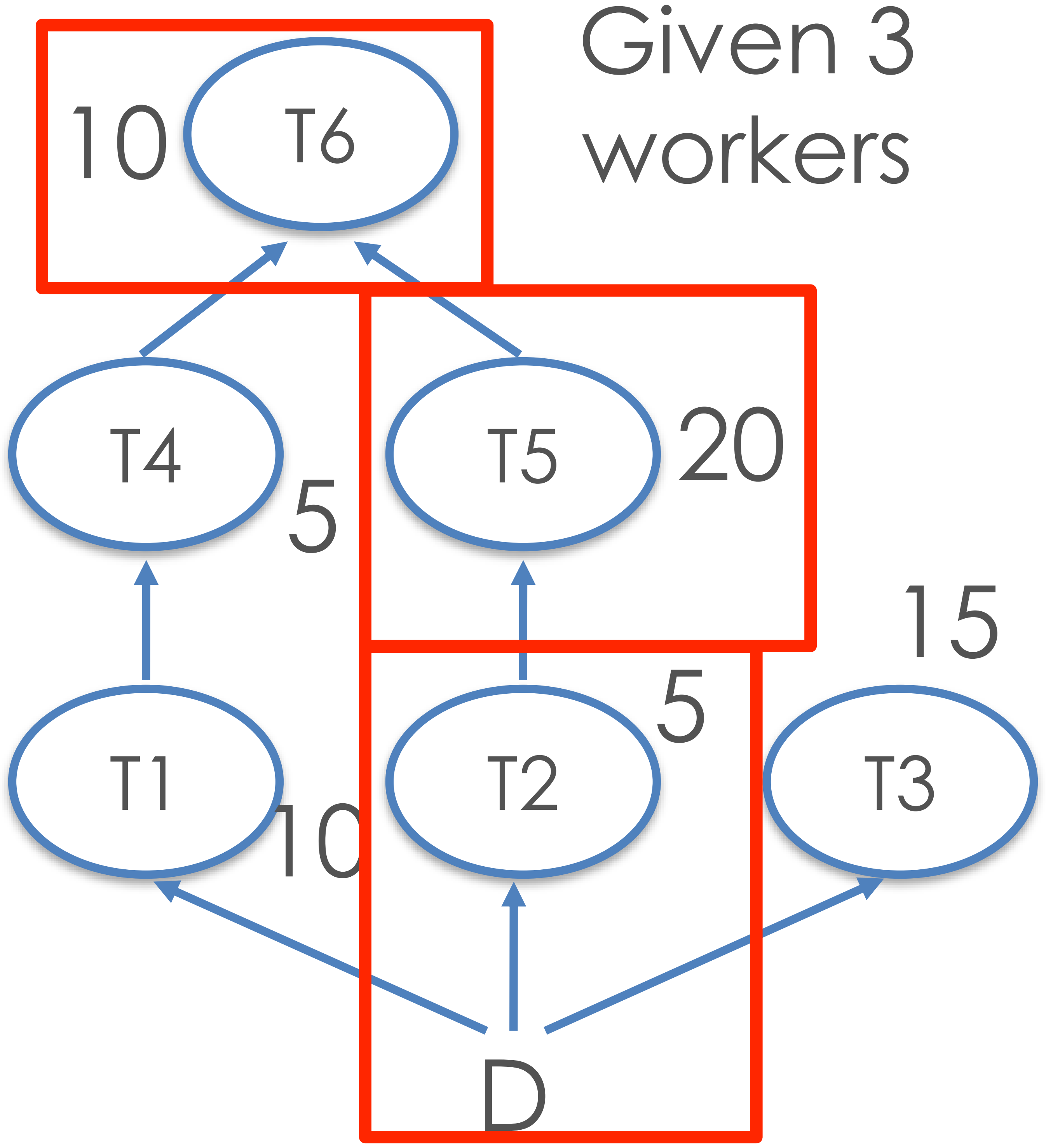


Idle Times in Task Parallelism

Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:

Given 3 workers



- In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- Possibility: A task-parallel scheduler can "release" a worker if it knows that will be idle till the end
- Can saves costs in cloud