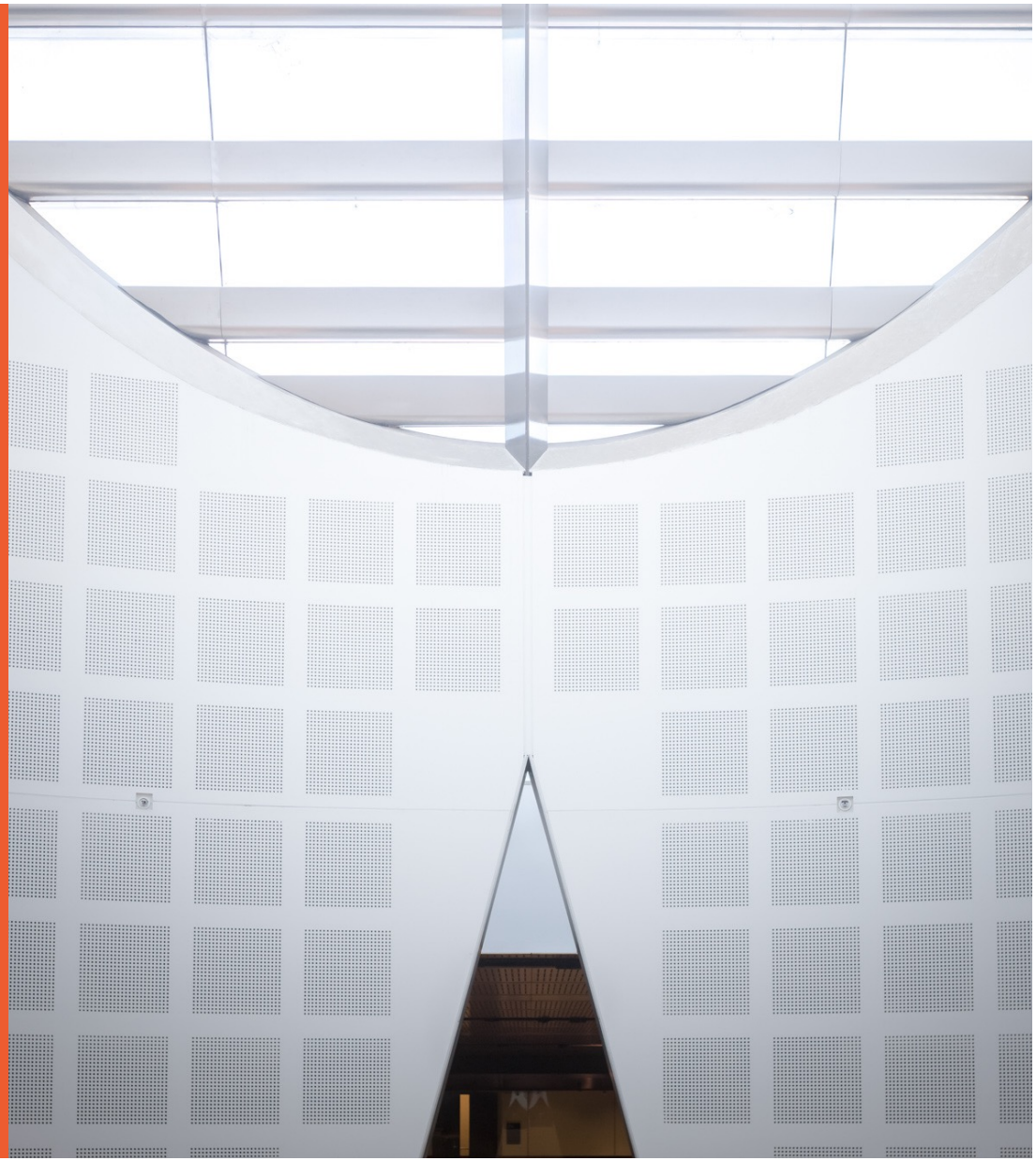
COMP5339: Data Engineering

W1: Introduction

Presented by
Uwe Roehm
School of Computer Science

THE UNIVERSITY OF SYDNEY

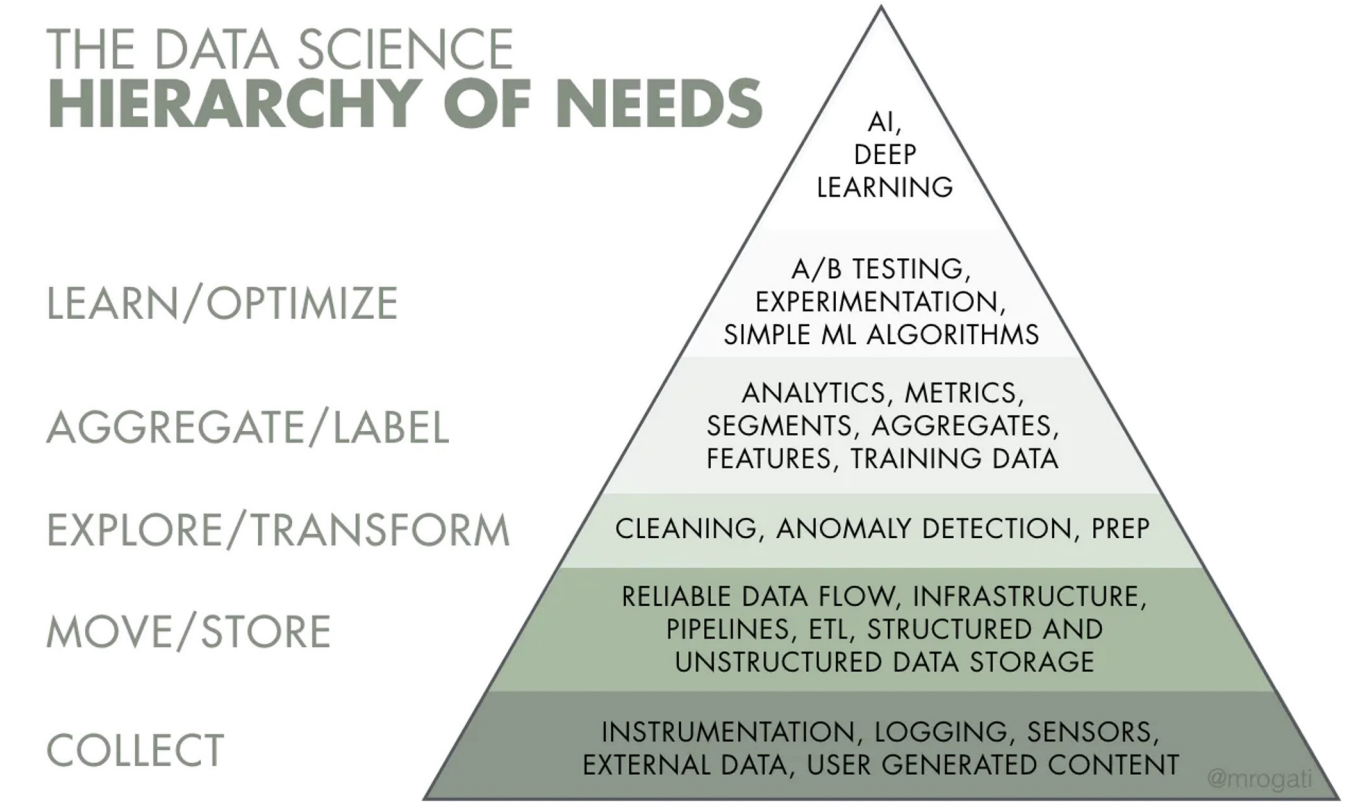


Data Science

"Many people imaging that data science is mostly machine learning […] . In fact, data science is mostly turning business problems into data problems and collecting data and cleaning data and formatting data, after which machine learning is almost an afterthought."

[Data Science from Scratch, Ch. 11]

Data Science – Hierarchy of Needs



Source: https://oriel.ly/pGg9U

Data Engineering

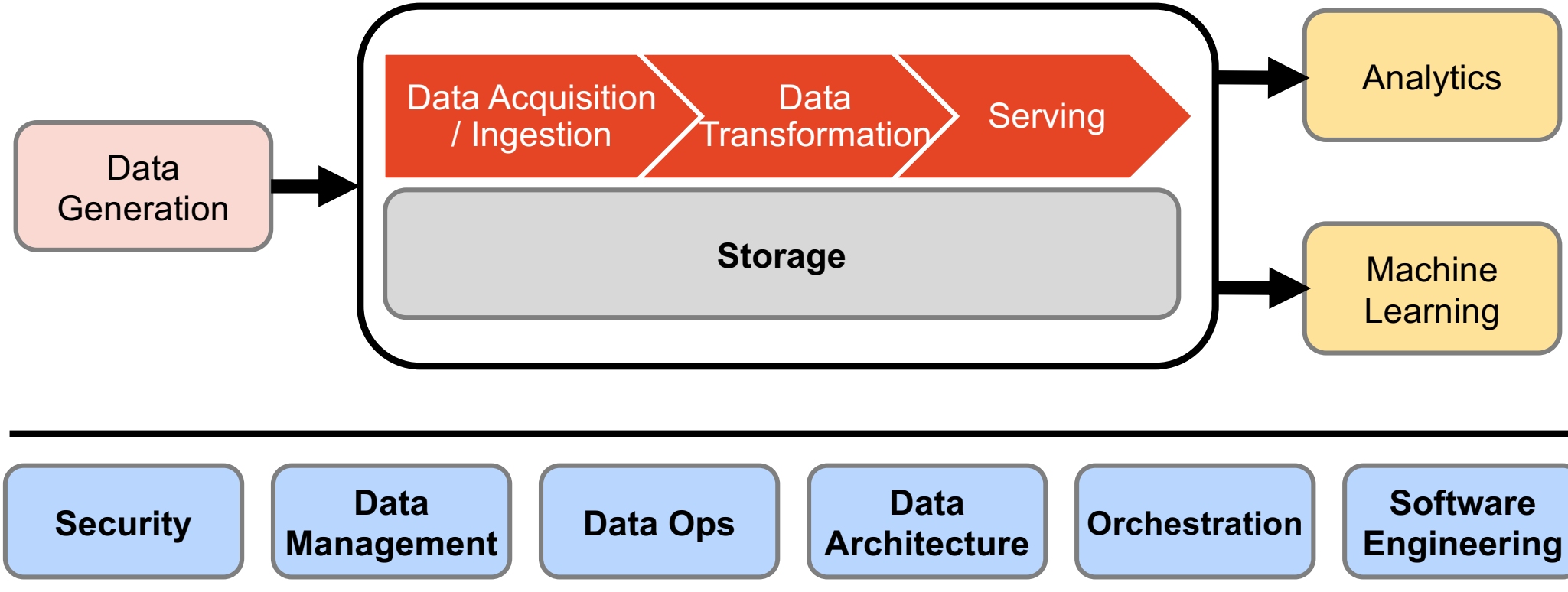
"Data engineering is the development, implementation, and maintenance of systems and processes that take in raw data and produce high-quality, consistent information that supports downstream use cases, such as analysis and machine learning. Data engineering is the intersection of security, data management, DataOps, data architecture, orchestration, and software engineering. A data engineer manages the data engineering lifecycle, beginning with getting data from source systems and ending with serving data for use cases, such as analysis or machine learning."

[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Evolution of Data Engineering

- Early days (1970 to 2000):
  From Databases to Data Warehouses to the Web
- Early 2000s: Cloud Computing, Cloud Storage, MapReduce
- Late 2000s to 2010s: Big Data Engineering
- The 2020s: Engineering for the Data Lifecycle

Data Engineering Lifecycle

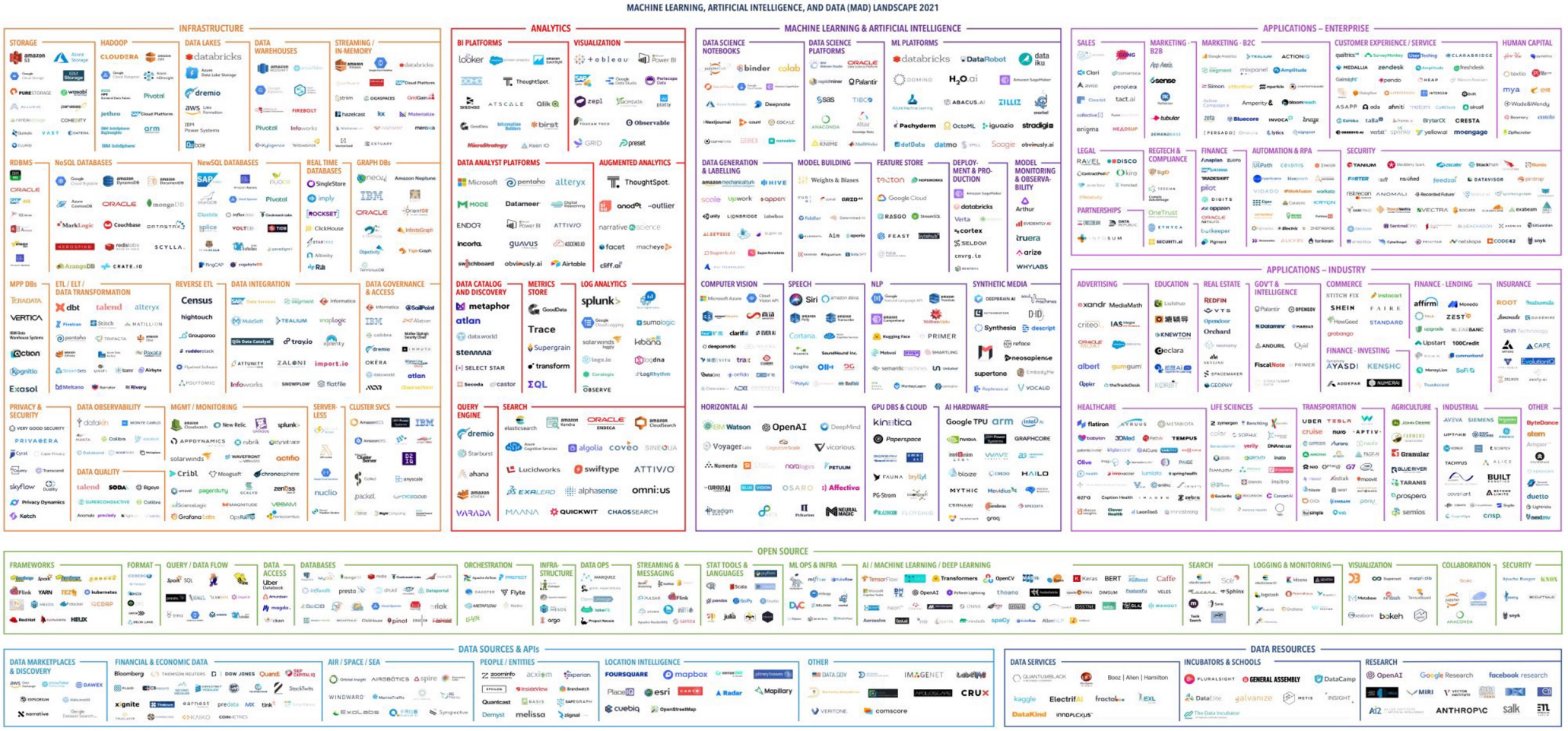


[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Approach in this Unit of Study

- We will take a rather data-centric approach and look at common use cases for data generation.
  - For example, working with existing databases, scraping data from the web or web services, working with unstructured data.
- For each case, we will look at the data engineering lifecycle including data ingestion, transformation and cleaning. Core to this will be to understand the underlying data model.
- Along the way, we will also cover the undercurrents such as data management, security, data quality, etc.
- Technologies used are taken as examples to learn the principles.

Matt Turck's Machine Learning, AI and Data (MAD) Landscape



Data Engineers

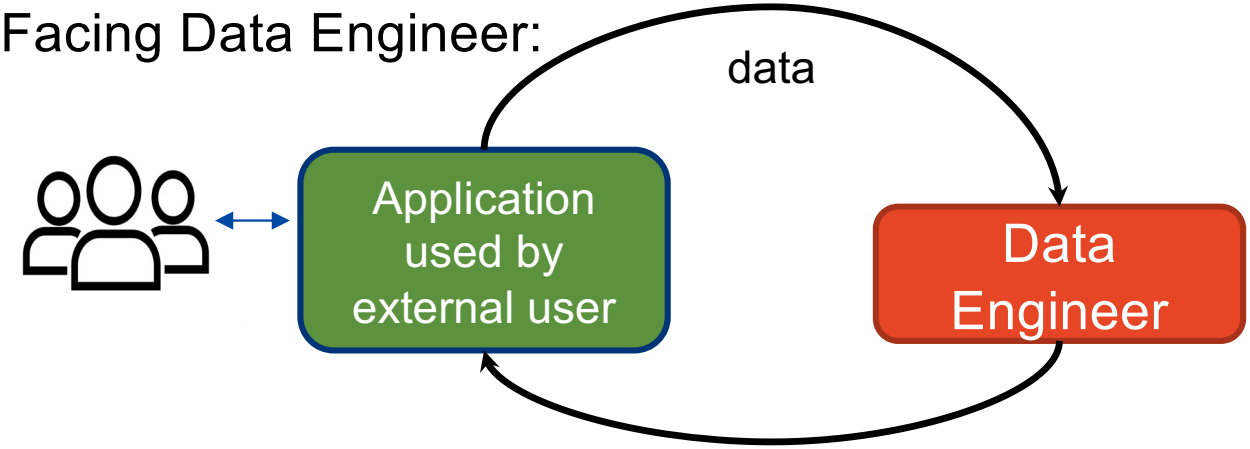
- Type A Data Engineers
  - *A* for **Abstraction:** Keep the data architecture as abstract and straightforward as possible, and manage the data engineering lifecycle by using off-the-shelf products, managed services and tools as much as possible.

- Type B Data Engineers
  - *B* for **Build:** Build data tools and systems that scale, and leverage a company's core competency to scale and lead with data; or build solutions for a use case so unique or mission-critical that custom data tools are required.

Internal-Facing vs. External-Facing Data Engineers

Internal-Facing Data Engineer:

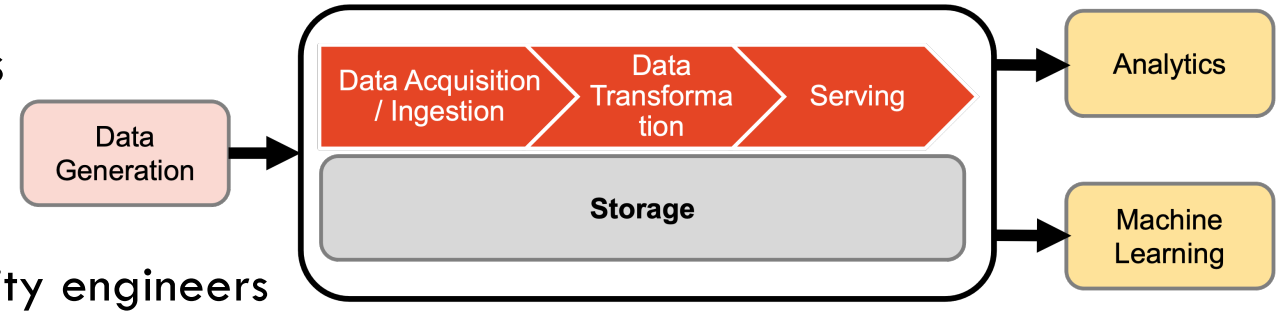
External-Facing Data Engineer:



[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Data Engineers at the nexus of various Roles and Stakeholders

- Upstream Data Producers
  - Software Engineers
  - Data Architects
  - DevOps and site-reliability engineers
- Downstream Data Consumers
  - Data Analysts
  - Data Scientists
  - Machine Learning Engineers and AI researchers
- Business Leadership
  - Project and Product Managers
  - Chief Information Officer (CIO), Chief Technology Officer (CTO), Chief-Data-Officer (CDO)



Unit Arrangements

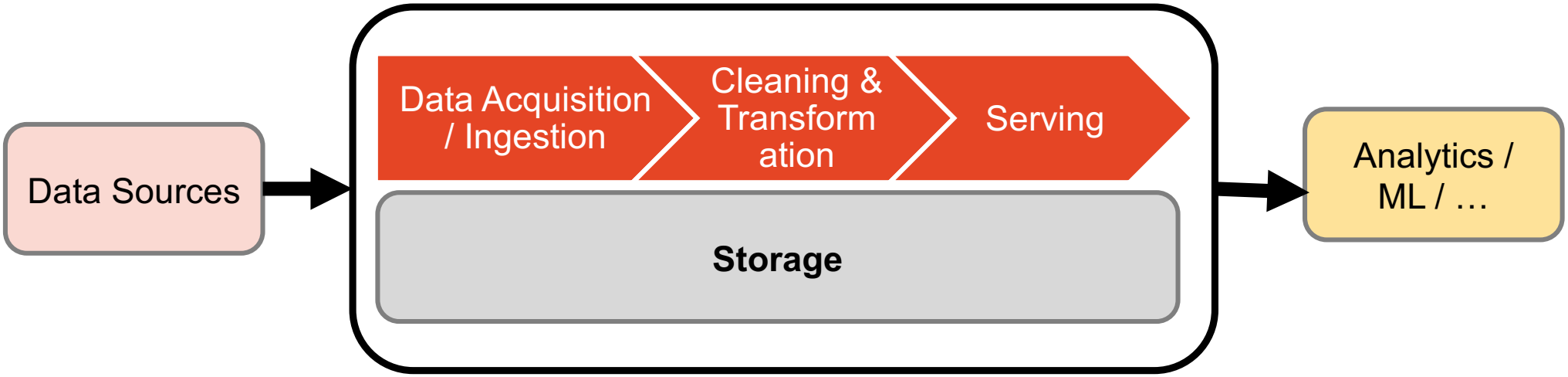
THE UNIVERSITY OF SYDNEY

Organisational Slides

Cf. separate slide deck **COMP5339_Organisation.pdf**

Data Pipeline

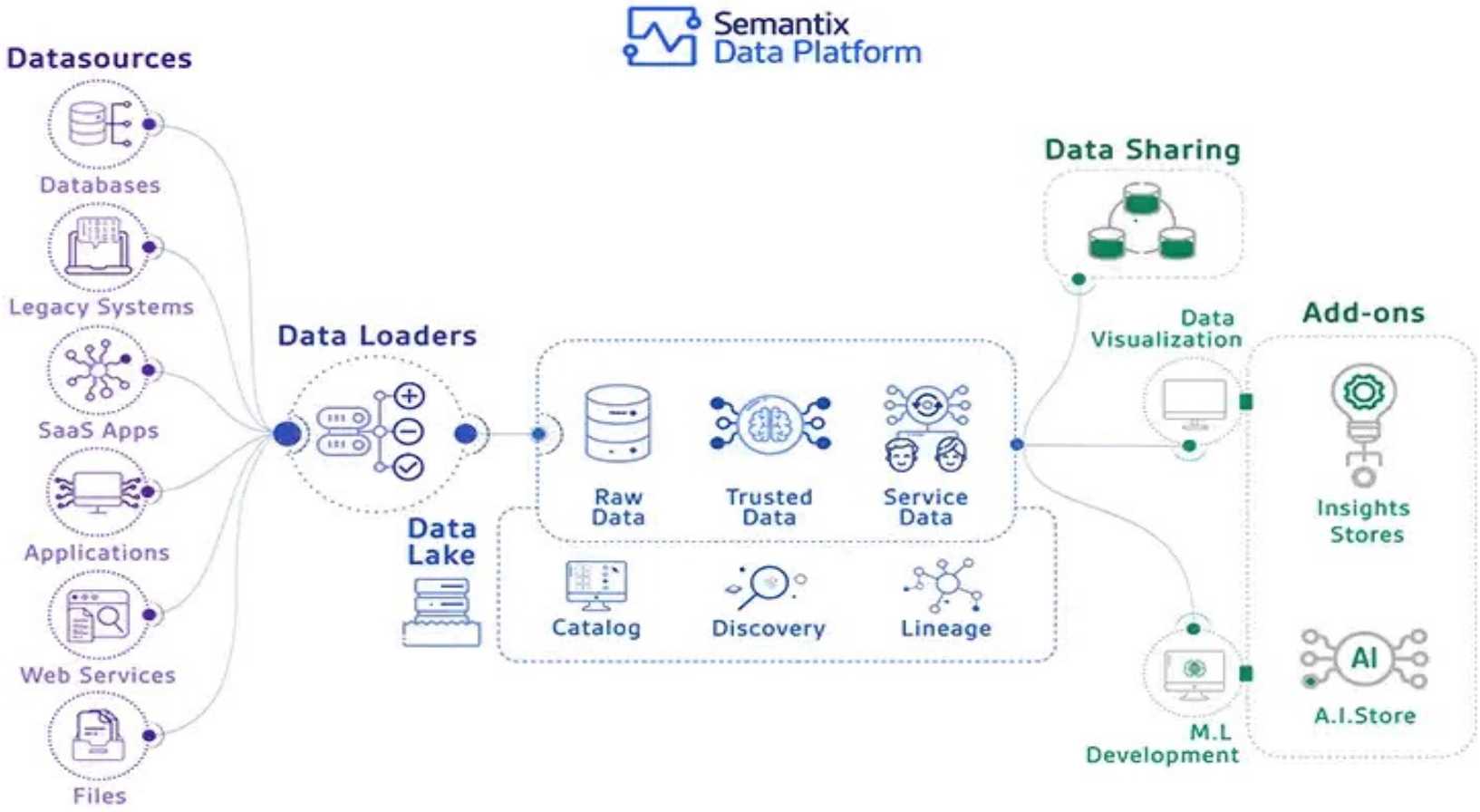
Typical Data Pipeline



"A data pipeline is the combination of architecture, systems, and processes that move data through the stages of the data engineering lifecycle."

[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Data Pipeline



Storage

- Choosing a storage solution is important for the success of a data pipeline
- Runs across the entire data engineering lifecycle, impacts all stages of a data pipeline
  - For example, cloud data warehouses can store data, process data in pipelines, and serve it to analysts;
  - Streaming frameworks such as Apache Kafka can function simultaneously as ingestion, storage, and query systems for messages…
- If various data models are combined, more than one storage solution might be needed.
  - E.g. structured data vs. unstructured data vs. graph data etc.
- Few data storage solutions function purely as storage, with many supporting complex transformation queries
  - For example, database systems or Amazon S3 Select.
  - In some cases, even basic machine learning functionality is supported.

Key Engineering Considerations re Storage

- What type of storage solution fits your use case best?
  - There is no one-size-fits-all universal storage recommendation. Every storage technology has its trade-offs.
- Is the storage system schema-agnostic (object storage)? Flexible schema? Enforced schema?
- Is this a pure storage solution (object storage), or does it support complex query patterns?
- Is this storage solution compatible with the architecture's required write and read speeds?
- Will this storage system handle the anticipated future scale?
- What are the required service-level agreement (SLA) for downstream users and processes?
- Are you capturing metadata about schema evolution, data flows, data lineage, and so forth?
- How are you handling regulatory compliance and data sovereignty? For example, can you store your data in certain geographical locations but not others?

Data Source Systems

- Possible sources: IoT devices, an application message queue, a website, an operational database….
- We need to understand the way source systems work, the way they generate data, the frequency and velocity of the data, and the variety of data they generate.
- Key engineering considerations
  - What are the essential characteristics of the data source? Is it an application? A transactional database? Is it a web source? A network of IoT devices?
  - At what rate is data generated? Respectively: How frequently should data be pulled from the source?
  - What is the type, variety, and schema of the ingested data?
  - Is there any personal or sensitive information involved? How is this managed? Is all of it really needed?
  - What data quality and level of consistency can data engineers expect from the source data?
  - If schema changes, how is this dealt with and communicated to downstream stakeholders?
  - How often do errors occur?
  - Will reading from a data source impact its performance?

Data Acquisition / Ingestion

- Data acquisition/ingestion are the most significant bottlenecks of a data pipeline
- Key considerations:
  - Can the ingested data be used in multiple use cases?
  - Batch data vs. Streaming data: Push vs. Pull? On-demand ingestion possible?
  - In what volume and velocity will the data typically arrive?
  - What format is the data in?
  - Is the source data in good shape for immediate downstream use or which data cleaning and transformation steps are needed?
  - Data transformation and cleaning needed immediately, or is first destination a staging area?
  - How frequently will I need to access the data?

Data Cleaning and Transformation

- Data needs to be changed from its original form into something useful for downstream use cases.
- Typically, the transformation stage is where data begins to create value for downstream user consumption.
- Key considerations
  - Which quality is the data, and what are the downstream quality requirements?
  - Can my systems handle the current format, or what data transformations are needed?
  - Is the transformation as simple and self-isolated as possible?
  - What's the cost and return on investment (ROI) of the transformation?
  - Data featurization for ML required?

Serving Data

- Data pipeline: ingest, store, clean, and transform data into coherent and useful structures
- How to get value from data?
  - "Getting value" means different things to different users.
- Data has value when it's used for practical purposes.

- Examples:
  - Data Analytics
  - Machine Learning
  - External services / APIs?

Process Data on the Command-Line

Unix…

- Unix command interpreter (aka the **shell**) allows automating data pipelines
- Unix operating system:
  - Development started in the 1970s at Bell Labs
  - AT&T then licensed Unix to outside partners
  - Various commercialization efforts (some very successful)
    - Sun Solaris, IBM AIX, Microsoft Xenix, HP-UX
    - Berkeley **BSD Unix**
- **Linux** started in 1991 as a separate Unix-like project by Linus Torvalds
  - Free Software Foundation refers to it as GNU/Linux
- Apple **macOS** is based on a Mach kernel with additional layers and tools derived from **BSD Unix**

Things to Keep in Mind

- Unix has a sophisticated command-line interface
  - but also many window manager interfaces
- Unix command line tools are **case-sensitive**
- Unix commands are **keyboard-centric** (every character counts!)
  - `ls` instead of "list" or "dir"
  - `chmod` instead of "change_mode"
- Unix **seldom asks** before executing something
  - so be very careful when issuing modification or deletion commands!
- Online help ('man-page') available to each command
  - `man` *`command`*
  - many Unix commands support '**-h**' or '**--help**' command line option for help

Commands to Navigate the Filesystem

`pwd`	display working directory
`cd <name>`	change directory
`cd`	change to home directory
`cd -`	change to previous directory
`cd /`	change to root directory
`ls [<name>]`	list (current) directory contents
`ls -al`	list everything with all details
`ls -R`	list directories recursively
`mkdir <name>`	make new directory
`mv <old> <new>`	rename / move a file or directory
`cp <source> <destination>`	copy a file

Looking at File Content

- `cat` *filename*
  - send file content to the standard output (usually the screen)
- `more` *filename* or `less` *filename*
  - display (text-)content of a file page-by-page (less is the more powerful command)
- `head` *filename*
  - output the (by default: 10) first lines of a file
- `tail` *filename*
  - output the (by default: 10) last lines of a file
- `wc` *filename*
  - word count of file content; shows: *num_line* *num_words* *num_characters*
- `sort` *filename*
  - output content of file sorted; many options on how to sort

Output Redirection

- Output of a Unix command is displayed by default on screen
  - also known as standard output or `stdout`
- `stdout` can be re-directed into a file
  - *command* > *filename*
- The standard error output, or `stderr` can also be redirected
  - command 2> filename

Combining Unix Tools: Piping

- Remember: in Unix everything is a file
  - In particular, the output of a Unix command is a 'stream' of data and can be directly used as input for a subsequent command
- No need to store the output of a command
  - Instead, the output of one command can be 'piped' into a subsequent Unix command using the | character

- Example:

```
cat filename | sort | head -3
```

pipe character

send the contents of `filename` to the standard input of the `sort` command

Combining Unix Tools: Piping

- Remember: in Unix everything is a file
  - In particular, the output of a Unix command is a 'stream' of data and can be directly used as input for a subsequent command
- No need to store the output of a command
  - Instead, the output of one command can be 'piped' into a subsequent Unix command using the | character

- Example:

```
cat filename | sort | head -3
```

read the file data from `cat`, `sort` it and output to `head` command

Spell Check in One Line of Shell!

```
cat textfile |
   tr 'A-Z' 'a-z' |       # convert upper to lower case
   tr -cs 'A-Za-z' '\n'| # one word per line, remove dups
   sort  |               # sort the file of words
   uniq  |               # remove duplicates
   # display words in textfile not in dict
   comm -23 - /usr/share/dict/words
```

Finding Data inside a File: grep or egrep

`grep` *pattern filename*

- Searches for patterns in file contents
- Displays matching lines
- Note: case sensitive

Pattern Matching: Regular Expressions

- Sequence of characters that define a search pattern
  - Special characters for wildcards, options, repetitions, conjunctions

- Wildcard: . matches any 1 char — gr.y
- Quantification:
  - ? 0 or 1 (optional) — colou?r
  - * 0 or more — ab*c
  - + 1 or more — ab+c
  - {n} n times
- Alternatives in […] — gr[ae]y
- Negation: ^ — gr[^bcdf-z]y
- start of line (^), end of line ($) — ^special$
- grouping — \([A-Z]+\)

Processing Content of Files: sed

- `sed` is a powerful stream editor for Unix
  - Typically, you pipe data 'through' `sed` for some automated changes
- Example: Text replacement

```
sed -e 's/from/to/g' filename
```

  - `s` stands for substitution
  - *`from`* is the word to be matched
  - *`to`* is the replacement string
  - `g` specifies to apply this to all occurrences on a line, not just the first.

```
echo "hello world" | sed -e 's/hello/bonjour/g' > output.txt
```

Processing Content of Files: awk

- AWK is a programming language for text processing and data extraction
  - AWK was created at Bell Labs in the 1970s, and its name is derived from the surnames of its authors—Alfred Aho, Peter Weinberger, and Brian Kernighan
- Very powerful pattern matching language, where code blocks can be executed for each match, and data be extracted into variables or send to output
  - Executes a sequence of ***pattern { action }*** on each line of the input
  - Lines are automatically separated into fields; $0 (line), $1, $2, … , $NF
  - Field separators can be specified
  - Special BEGIN and END 'patterns' to execute at start or end of a file

Review

THE UNIVERSITY OF SYDNEY

Review

- What is data engineering?
- Data pipeline
- Process data on the command-line