Fakultas Informatika
School of Computing
Telkom University

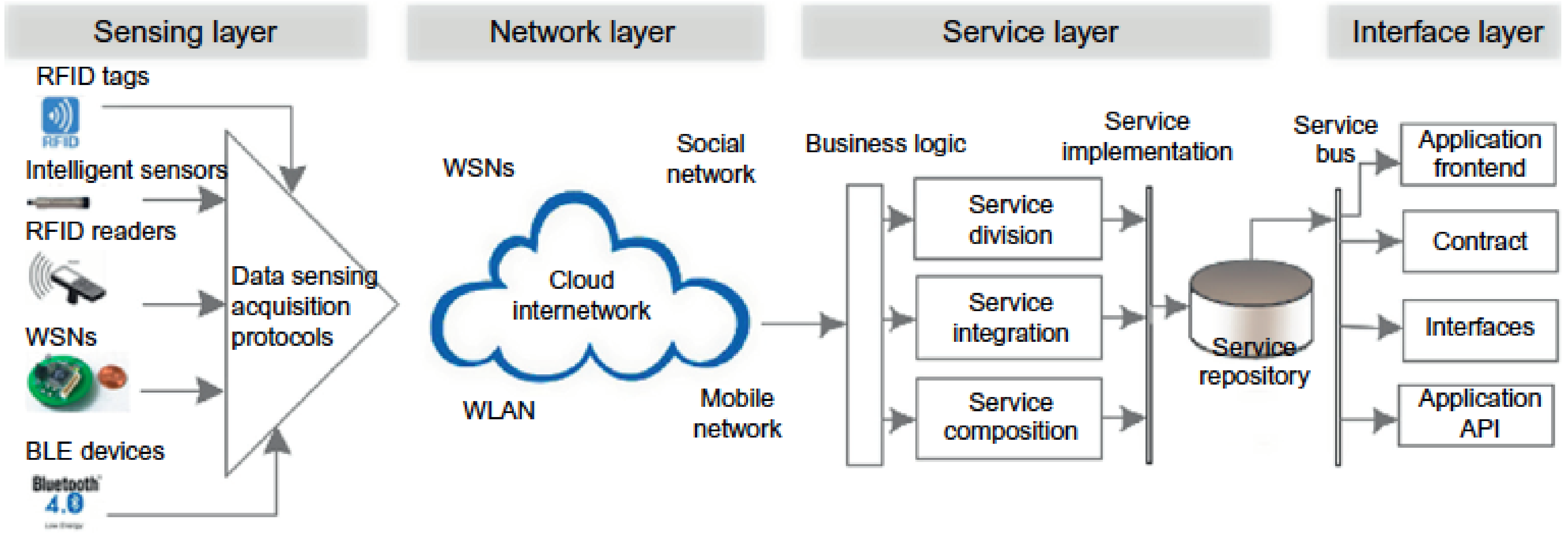
Internet of Things

MATERI 7:
Data Analytic for IoT

What we learn today ...

- Overview
- Basic data analytics in IoT
- Top-level cloud pipeline
  - Rules engines
  - Ingestion – streaming, processing, and data lakes
  - Complex event processing
  - Lambda architecture

Which layer is the IoT Data Analytic located on?



Data Analytic for IoT

- The value of an IoT system is not a single sensor event, or a million sensor events archived away.
- A significant value of IoT is in the interpretation and decision made of that data.

Data Analytic for IoT

- Data also may need to be interpreted and analyzed in real time as a streaming dataflow, or it may be archived and retrieved for deep analytics in the cloud.
- Analytics for the IoT segment deal with:
  - **Structured data** (SQL storage), a predictable format of data.
  - **Unstructured data** (raw video data or signals), a high degree of randomness and variance.
  - **Semi-structured** (Twitter feeds), some degree of variance and randomness in form.

ANY
QUESTIONS
?

Basic Data Analytic for IoT

- Data analytics intend to find events, usually in a streaming series of data.
- There are multiple types of events and roles that a real-time streaming analysis machine must provide.

Basic Data Analytic for IoT

- **Preprocessing**: Filter out events of little interest, denaturing, feature extraction, segmentation, transform data to a more suitable form (although data lakes prefer no immediate transformation), adding attributes to data such as a tag (data lakes do need tags).
- **Alerting**: Inspect data; if it exceeds some boundary condition, then raise an alert. The simplest example is if the temperature rises above a set limit on a sensor.

Basic Data Analytic for IoT

- **Windowing**: A sliding window of events is created that only draws rules upon that window. Windows can be based on time (for example, one hour), or length (2000 sensor samples). They can be sliding windows (for example, inspect only the 10 latest sensor events and produce a result whenever a new event arises), or batch windows (for example, produce an event only at the end of the window). Windowing is good for rules and for counting events. One could look for the number of temperature spikes in the last hour and resolve that a defect will occur on some machine.

Basic Data Analytic for IoT

- **Joins**: Combine multiple data streams into a new single stream. A scenario where this applies is a logistics example. Say a shipping company tracks their shipments with assets tracking beacons and that their fleet of trucks, planes, and facilities have geolocation information streaming as well. There are initially two streams of data: one for the package, and one for a given truck. When a truck picks up a package, those two streams become joined.

ANY
QUESTIONS
?

Basic Data Analytic for IoT

- **Errors**: Millions of sensors will generate missing data, garbled data, and data that is out of sequence. This is important in the IoT case with multiple streams of asynchronous and independent data. For example, data may be lost in a cellular WAN if a vehicle enters an underground parking garage. This analytic pattern correlates data within its own stream to attempt to find these error conditions.

Basic Data Analytic for IoT

- **Databases**: The analytics package will need to interact with some data warehouse. For example, if data is streaming in from a number of sensors looking, or in particular, when Bluetooth asset tags if an item is stolen or lost, a database of missing tag IDs would be referenced from all the gateways streaming in tag IDs to the system.

Basic Data Analytic for IoT

- **Temporal events and patterns**: This is most often used with the window pattern mentioned previously. Here, a series or sequence of events constitutes a pattern of interest. One can think of this as a state machine. Say we are monitoring the health of a machine based on temperature, vibrations, and noise. A temporal event sequence could be as follows:
  - Detect if the temperature exceeds 100° C
  - Then detect if vibrations exceed 1 m/s
  - Next, detect if the machine is emitting noise at 110 dB
  - If those events take place in that sequence, only then raise an alert

ANY QUESTIONS ?

Basic Data Analytic for IoT

- **Tracking**: Tracking involves when or where something exists, an event occurred, or when something doesn't exist where it should have. A very basic example is geolocation of service trucks where a company may need to know exactly where a truck is, and when it was last there. This has application in agriculture, human movement, tracking patients, tracking high-value assets, luggage systems, smart city garbage, snow removal, and so on.

Basic Data Analytic for IoT

- **Trends**: This pattern is particularly useful for predictive maintenance. Here, a rule is designed to detect an event based on time-correlated series data. This is similar to temporal events, but differs in the sense that temporal events have no notion of time, only sequence order. This model uses time as a dimension in the process. A running history of time-correlated data could be used to find patterns like a livestock sensor in farming. Here, a head of cattle may wear a sensor that detects the animal movement and temperature. An event sequence can be constructed to see if the cattle moved in the last day. If there was no movement, the cattle may be sick or dead.

Basic Data Analytic for IoT

- **Batch queries**: Batch processing typically is more comprehensive and deeper than real-time stream processing. A well-designed streaming platform can fork analysis and call into a batch processing system. This will be talked about later in the form of Lambda processing.
- **Models and training**: The first-level model described previously may, in fact, be an inference engine for a machine learning system. These machine learning tools are built on trained models that can be used for in-flight, real-time analysis.

ANY
QUESTIONS
?

Basic Data Analytic for IoT

- **Deep analytics pathway**: In real-time processing, we make a decision on the fly that some event has occurred. Whether or not that event really should signal an alarm may require further processing that will not operate in real time. This works because these events should be rare, and pass down information to a detailed analysis engine, while new events streaming in real time should be designed within a system.

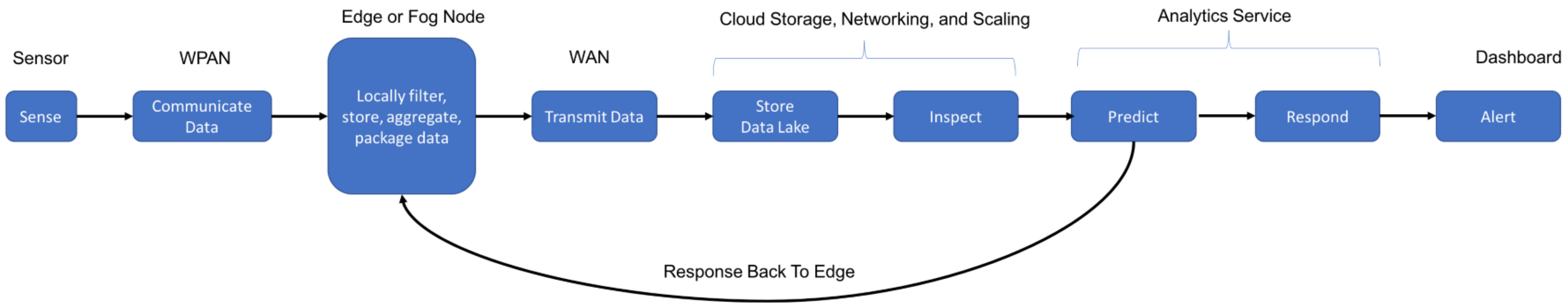
Basic Data Analytic for IoT

- **Signaling**: It is often the case that an action needs to propagate back to the edge and sensor. A typical case is factory automation and safety. For example, if the temperature rises beyond a certain limit on a machine, log the event, but also send a signal back to the edge device to slow the machine down. The system must be able to be bidirectional in communication.
- **Control**: Finally, we need a way to control these analysis tools. Whether that is starting, stopping, reporting, logging, or debugging, facilities need to be in place to manage this system.

ANY
QUESTIONS
?

Top-level cloud pipeline

- The following diagram is a typical flow of data from a sensor to a dashboard.



Top-level cloud pipeline

- Data will transit through several mediums (WPAN links, broadband, cloud storage in the form of a data lake, and so on).
- When we consider the following architectures to build a cloud analytics solution, we have to consider the effects of scaling.

Top-level cloud pipeline

The analytics (predict-respond) portion of the cloud can take on several forms:

- **Rules engines**: These simply define an action and produce an outcome.
- **Complex event processing**: This is based on queries like SQL and written in that higher level language. It is based on event processing, and is tuned for low latency.
- **Lambda architecture**: This model attempts to balance throughput and latency by performing batch processing and stream processing in parallel on massive data sets.

Top-level cloud pipeline

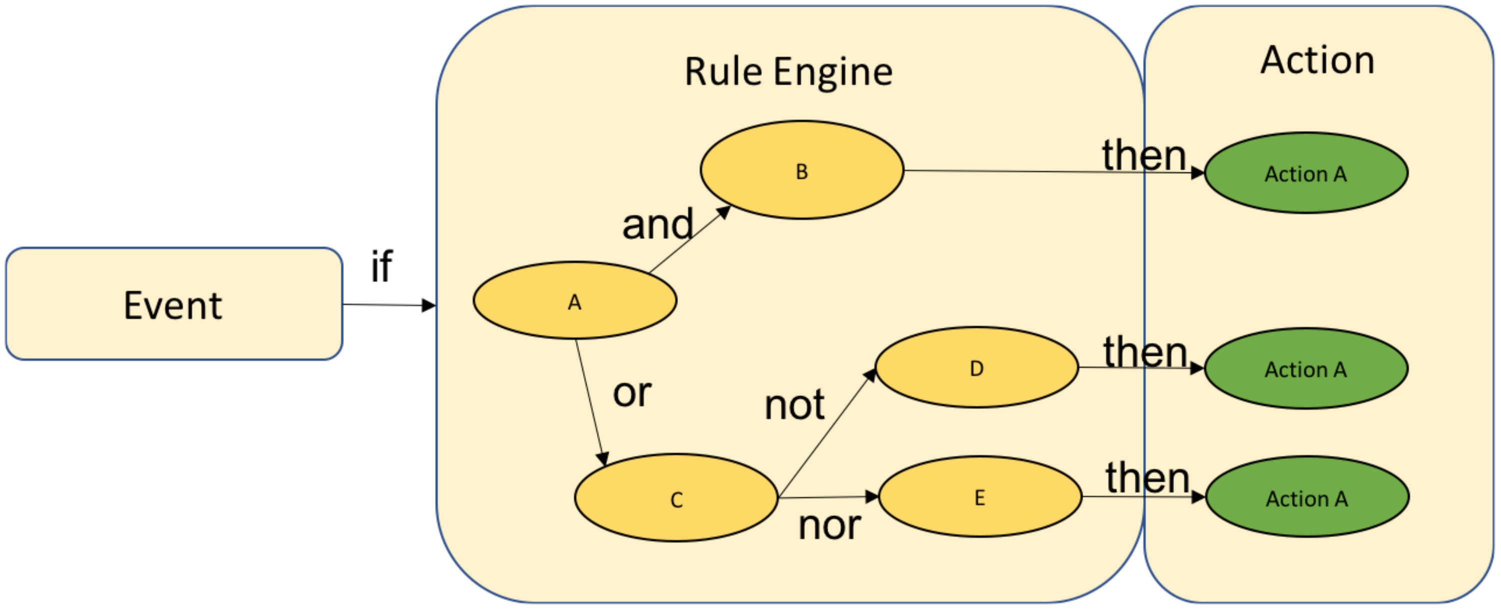
- **Stream processing**: These are where events like sensor readings are injected into the stream processor. The processing path is a graph where nodes in the graph represent operators, and send events to other operators. The nodes contain the code for that portion of the processing, and a path to connect to the next node in the graph. This graph can be replicated and executed in parallel on a cluster so it is amenable to scaling up to hundreds of machines.

ANY
QUESTIONS
?

Rules engines

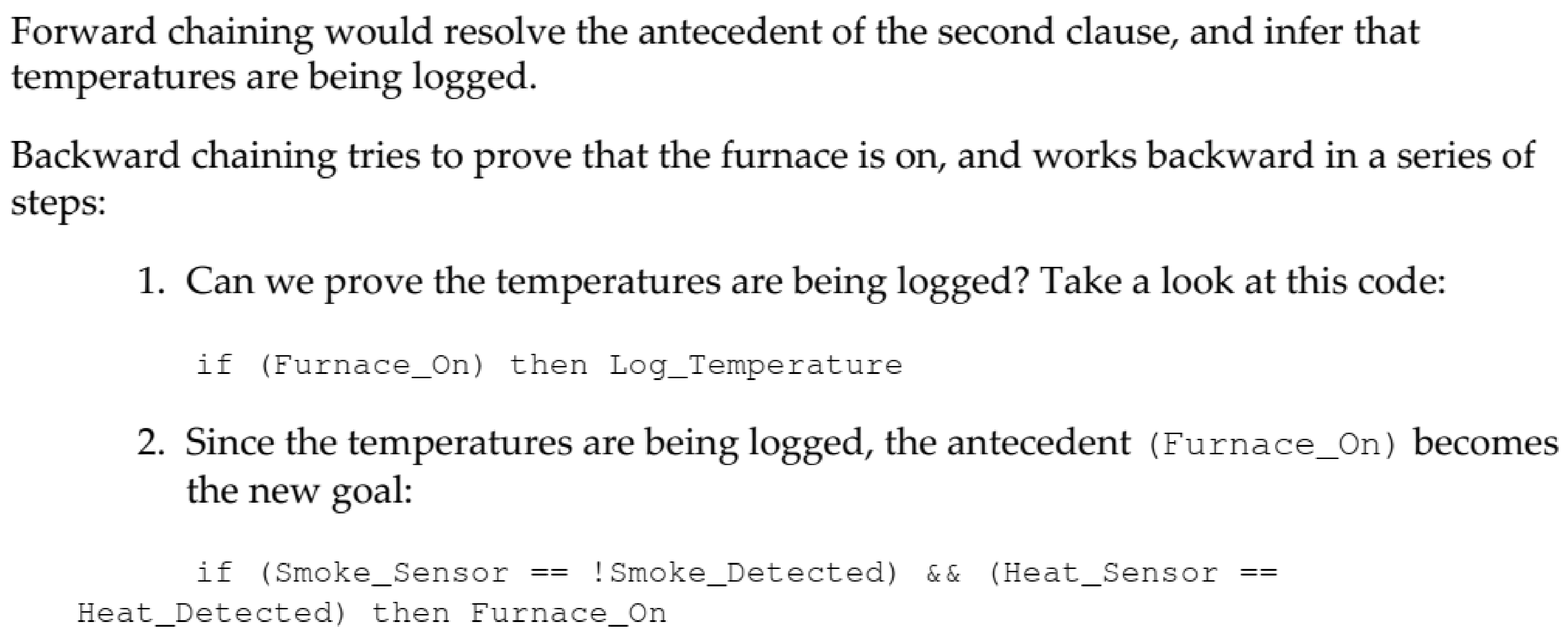
- A rule engine is simply a software construct that executes actions on events. For example, if the humidity in a room exceeds 50%, send an SMS message to the owner. These are also called Business Rule Management Systems (BRMS).
- Rules engines may or may not have state and be called **stateful**. That is, it may have a history of the event, and take different actions depending on the order, the amount, or the patterns of events as they occurred historically.

Rules engines



Rules engines

The following pseudocode demonstrates a simple rule engine:



Forward chaining would resolve the antecedent of the second clause, and infer that temperatures are being logged.

Backward chaining tries to prove that the furnace is on, and works backward in a series of steps:

1. Can we prove the temperatures are being logged? Take a look at this code:

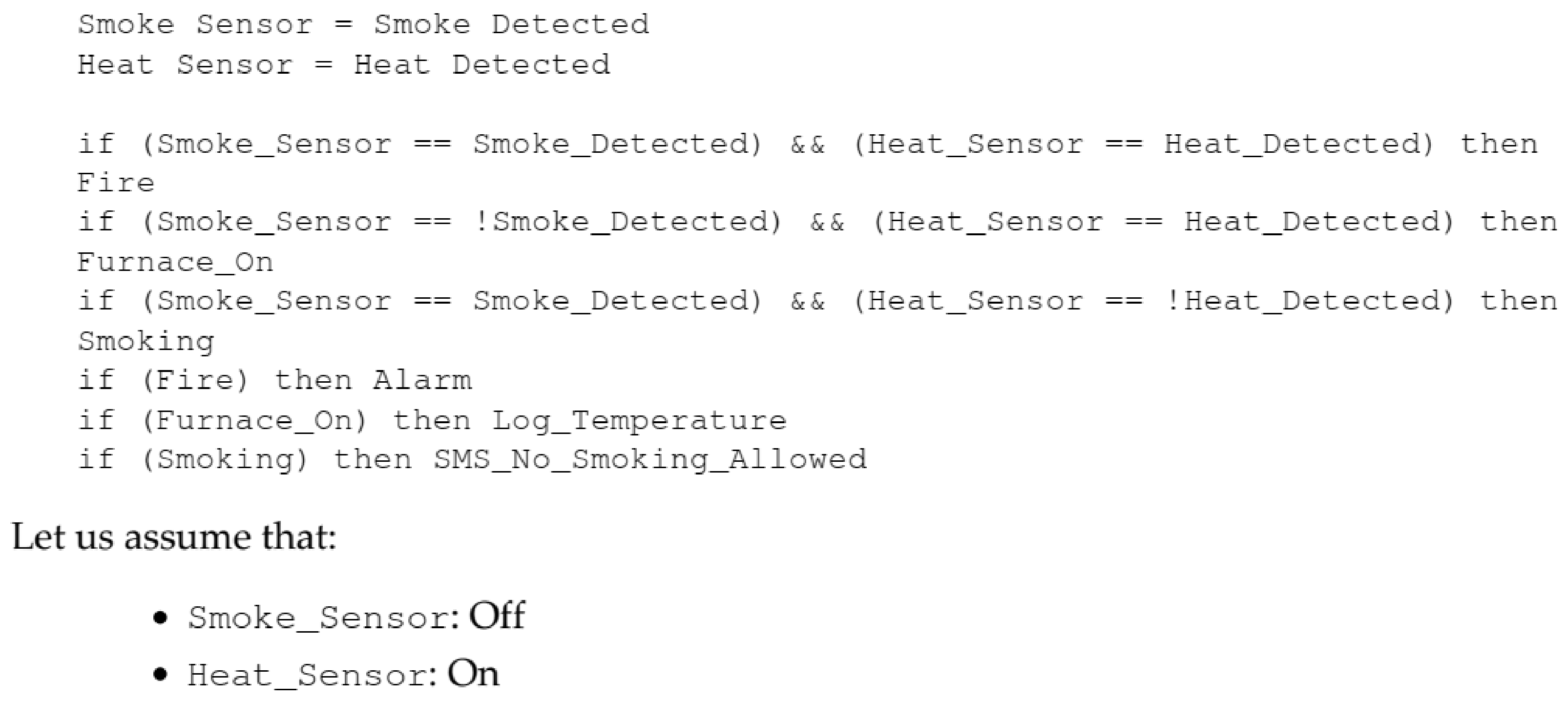
```
if (Furnace_On) then Log_Temperature
```

2. Since the temperatures are being logged, the antecedent `(Furnace_On)` becomes the new goal:

```
    if (Smoke_Sensor == !Smoke_Detected) && (Heat_Sensor ==
Heat_Detected) then Furnace_On
```

Rules engines

The following pseudocode demonstrates a simple rule engine:



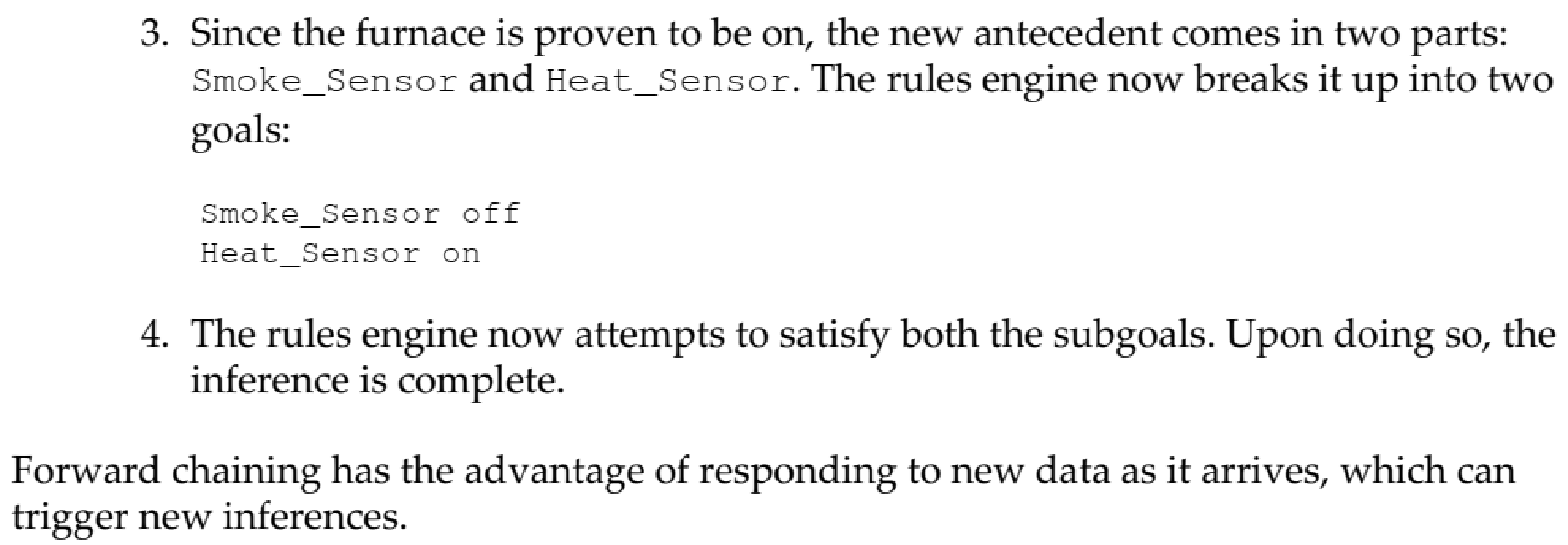
```
Smoke Sensor = Smoke Detected
Heat Sensor = Heat Detected

if (Smoke_Sensor == Smoke_Detected) && (Heat_Sensor == Heat_Detected) then
Fire
if (Smoke_Sensor == !Smoke_Detected) && (Heat_Sensor == Heat_Detected) then
Furnace_On
if (Smoke_Sensor == Smoke_Detected) && (Heat_Sensor == !Heat_Detected) then
Smoking
if (Fire) then Alarm
if (Furnace_On) then Log_Temperature
if (Smoking) then SMS_No_Smoking_Allowed
```

Let us assume that:

- `Smoke_Sensor`: Off
- `Heat_Sensor`: On

Rules engines

The following pseudocode demonstrates a simple rule engine:



3. Since the furnace is proven to be on, the new antecedent comes in two parts: `Smoke_Sensor` and `Heat_Sensor`. The rules engine now breaks it up into two goals:

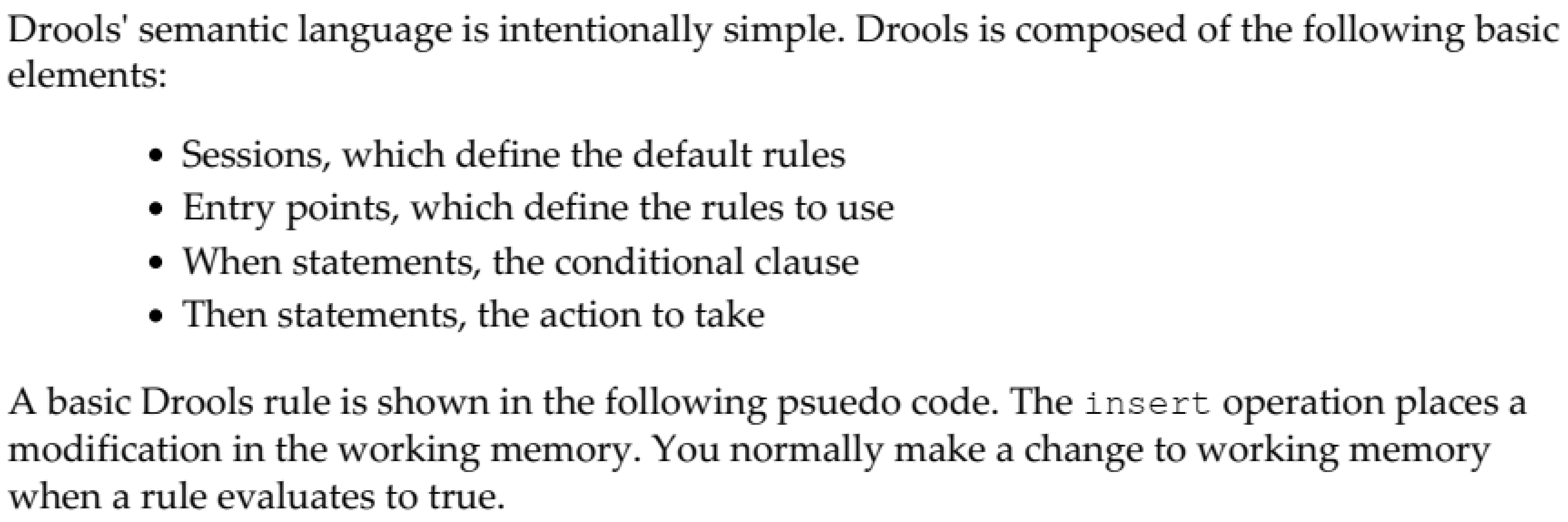
```
Smoke_Sensor off
Heat_Sensor on
```

4. The rules engine now attempts to satisfy both the subgoals. Upon doing so, the inference is complete.

Forward chaining has the advantage of responding to new data as it arrives, which can trigger new inferences.

Rules engines

The following pseudocode demonstrates a simple rule engine:



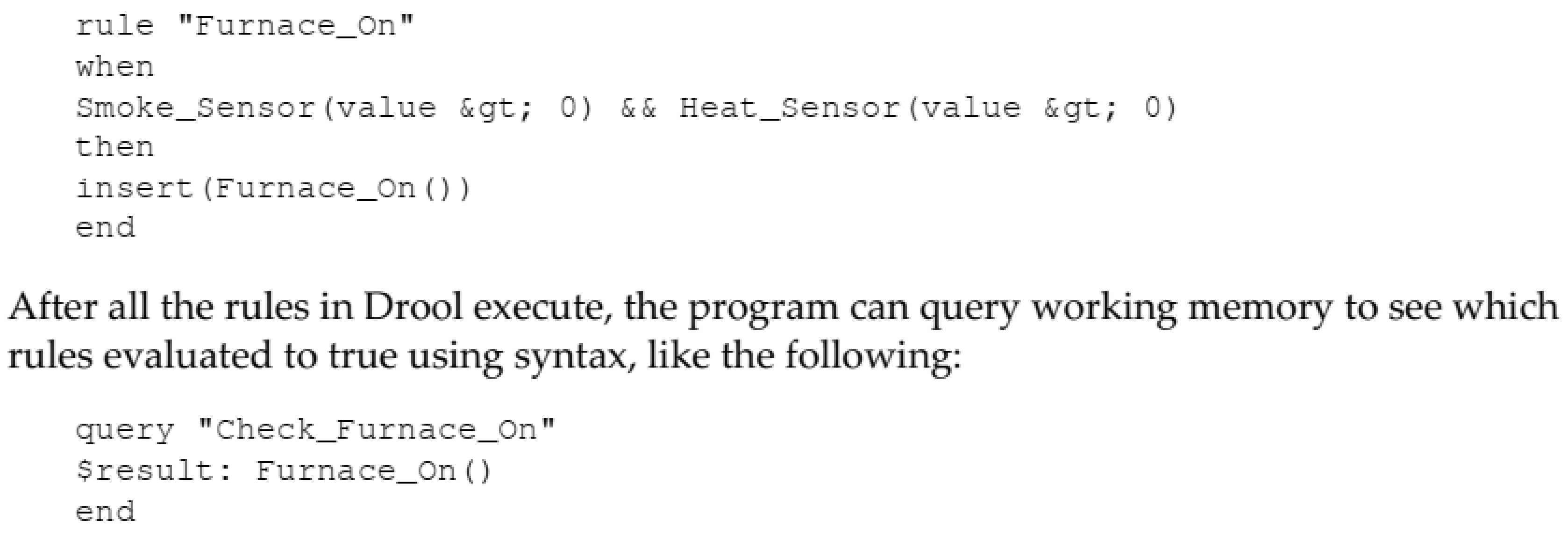
Drools' semantic language is intentionally simple. Drools is composed of the following basic elements:

- Sessions, which define the default rules
- Entry points, which define the rules to use
- When statements, the conditional clause
- Then statements, the action to take

A basic Drools rule is shown in the following psuedo code. The `insert` operation places a modification in the working memory. You normally make a change to working memory when a rule evaluates to true.

Rules engines

The following pseudocode demonstrates a simple rule engine:



```
rule "Furnace_On"
when
Smoke_Sensor(value &gt; 0) && Heat_Sensor(value &gt; 0)
then
insert(Furnace_On())
end
```

After all the rules in Drool execute, the program can query working memory to see which rules evaluated to true using syntax, like the following:

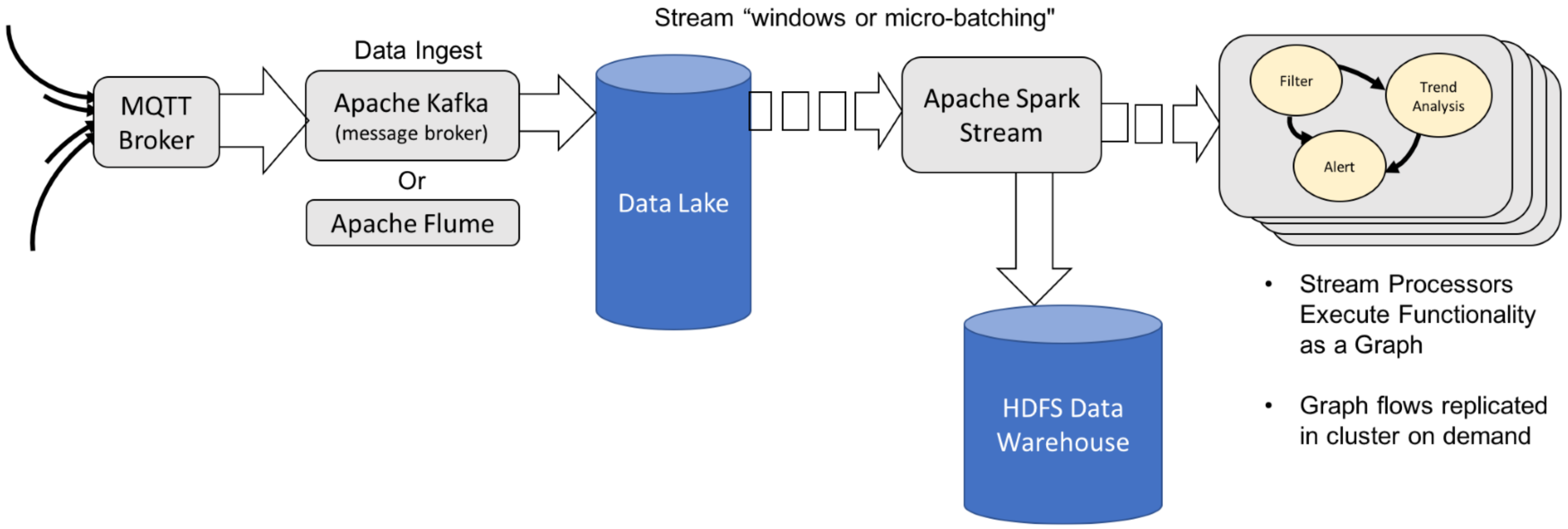
```
query "Check_Furnace_On"
$result: Furnace_On()
end
```

ANY
QUESTIONS
?

Ingestion – streaming, processing, and data lakes

- An IoT device is usually associated with some sensor or a device whose purpose is to measure or monitor the physical world. It does so asynchronously with respect to the rest of the IoT technology stack.
- That is, a sensor is always attempting to broadcast data, whether or not a cloud or fog node is listening. This is important, because the value of a corporation is in the data. Even if most of the data produced is redundant, there is always the opportunity that a significant event can occur.
- This is the data stream.

Ingestion – streaming, processing, and data lakes



- Stream Processors Execute Functionality as a Graph
- Graph flows replicated in cluster on demand

ANY
QUESTIONS
?

Complex event processing

- **Complex event processing** (CEP) is another analytic engine that is often used for pattern detection. A method capable of analyzing a live feed of streaming data in near real time.
- As hundreds and thousands of events enter the system, they are reduced and distilled into higher-level events. These are more abstract than raw sensor data.
- CEP engines have the advantage of a fast turnaround time in real-time analysis over a stream processor. A stream processor can resolve an event in the millisecond time frame.

Lambda architecture

- A Lambda architecture attempts to balance latency with throughput. Essentially, it mixes batch processing with stream processing.
- There are three layers of the topology:
  - **Batch layer**: The batch layer is usually based on Hadoop clusters. The batch layer is significantly slower in processing than the stream layer. By sacrificing latency, it maximizes throughput and accuracy.
  - **Speed layer**: This is the real-time in-memory data stream. The data can be erroneous, missing, and out of order. Apache Spark, as we have seen, is very good at providing a stream processing engine.
  - **Service layer**: Service layer is where the recombination of batch and stream results are stored, analyzed, and visualized.

ANY
QUESTIONS
?

Fakultas Informatika
School of Computing
Telkom University

THANK YOU