# 3 – Founderism, AI, and Coding

**Capitalism for Founders 101**
Thinking like an Entreprenuer

**Coding**
Going deeper

**AI**
Lets get the conversation started

# Are You on Github?

- Fill out Survey so we have all github repos
- We will be taking a look as you work and giving feedback through github!
- Github is a coders Instagram! You don’t exist if you’re not on it! :-D

# Founderism

The mentality and culture that produces founders

# Capitalism is like Farts

- No one really likes it
- Seems like we have to have it to survive
- We all kind of have to deal with it
- In practice, We kind of pretend that its ill effects doesn't exist (*In Practice*)

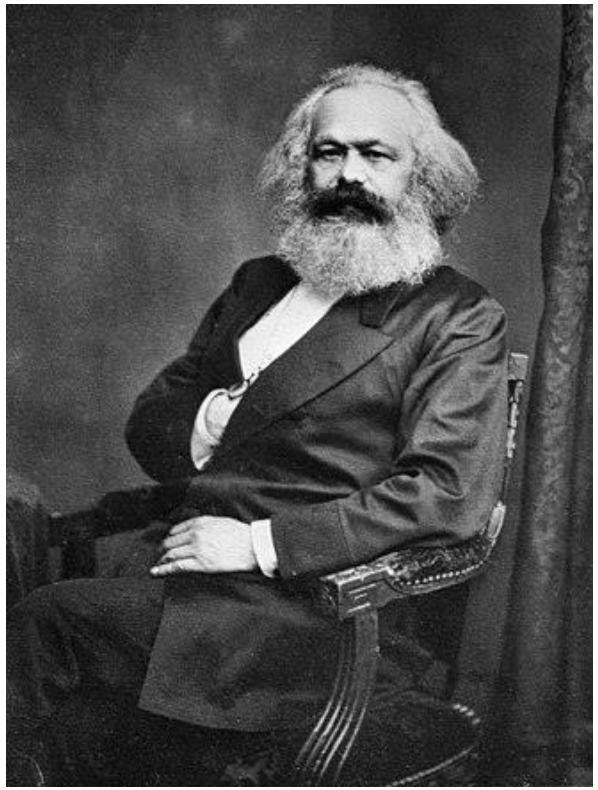
# Human Nature

- The have nots seek to have

- Those who have seek to keep

# Marx was on to something

- Karl Marx
  - German philosopher, critic of political economy, economist, historian, sociologist, political theorist, journalist and socialist revolutionary.
- Known as revolutionary, but really an academician and thinker
- Brilliant **Theories**
  - Not all, but many
- One very relevant to understanding the practice of our society…



# Bourgeoisie vs Proletariat

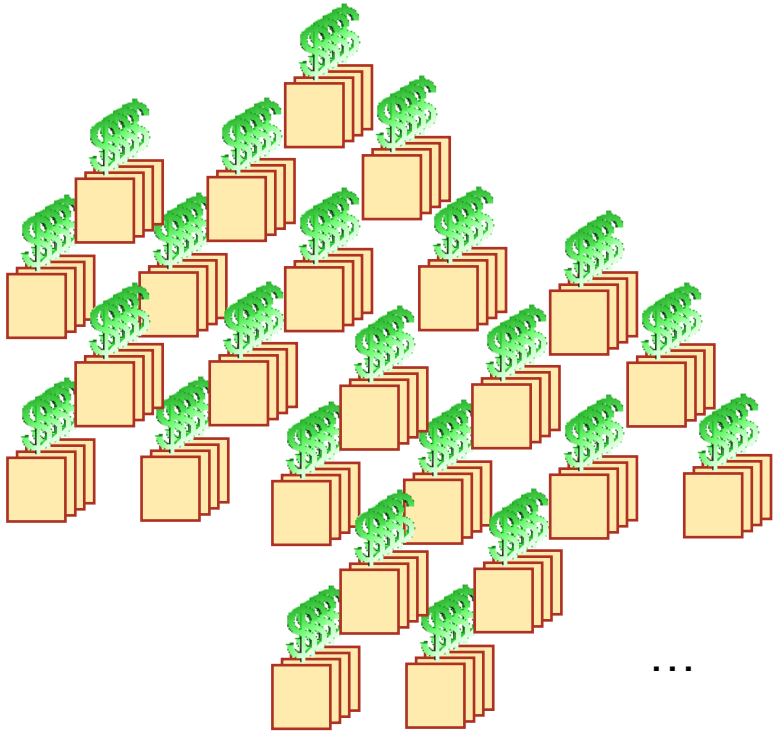
- Proletariat
  - the term proletariat designated the class of wage workers who were engaged in industrial production and whose chief source of income was derived from the **sale of their labour power**.
- Working at a company

- Bourgeoisie
  - the bourgeoisie is the social class that came to **own the means of production** during modern industrialization and whose societal concerns are the value of property and the preservation of capital.
- Owning a company

# Scaling

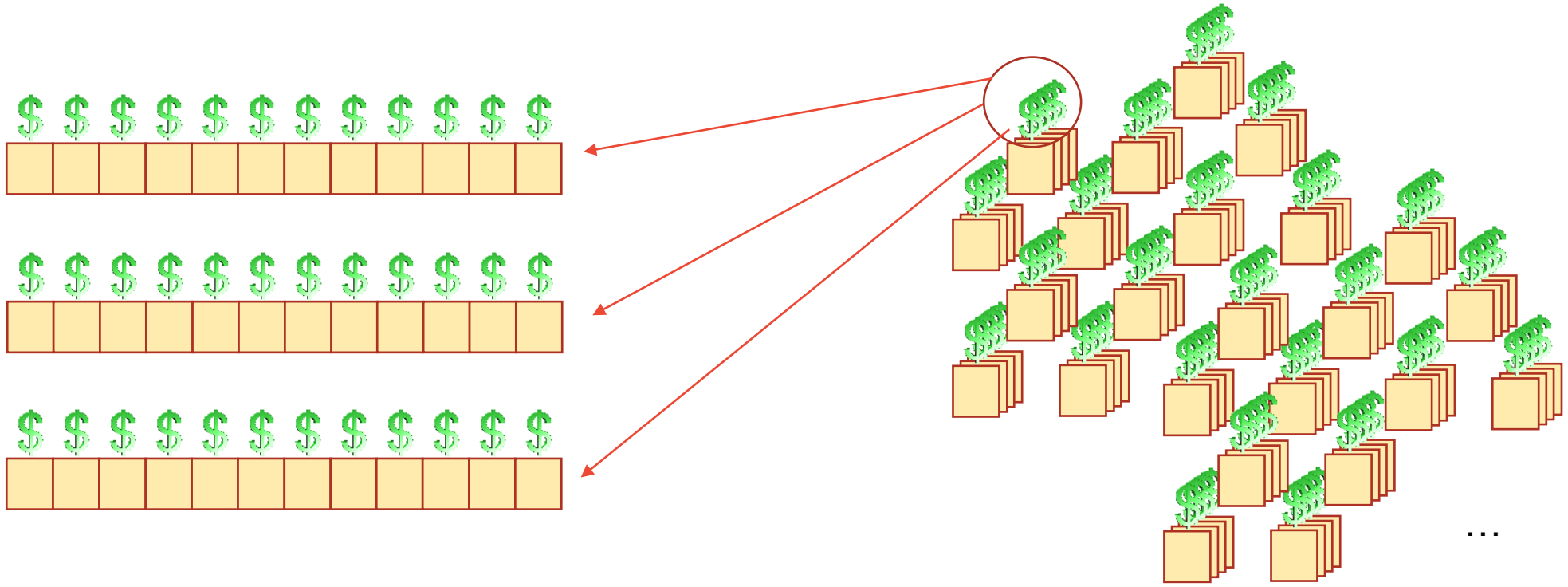
- Selling Time

- Selling Things



# Scaling

- Selling Time
- Selling Things

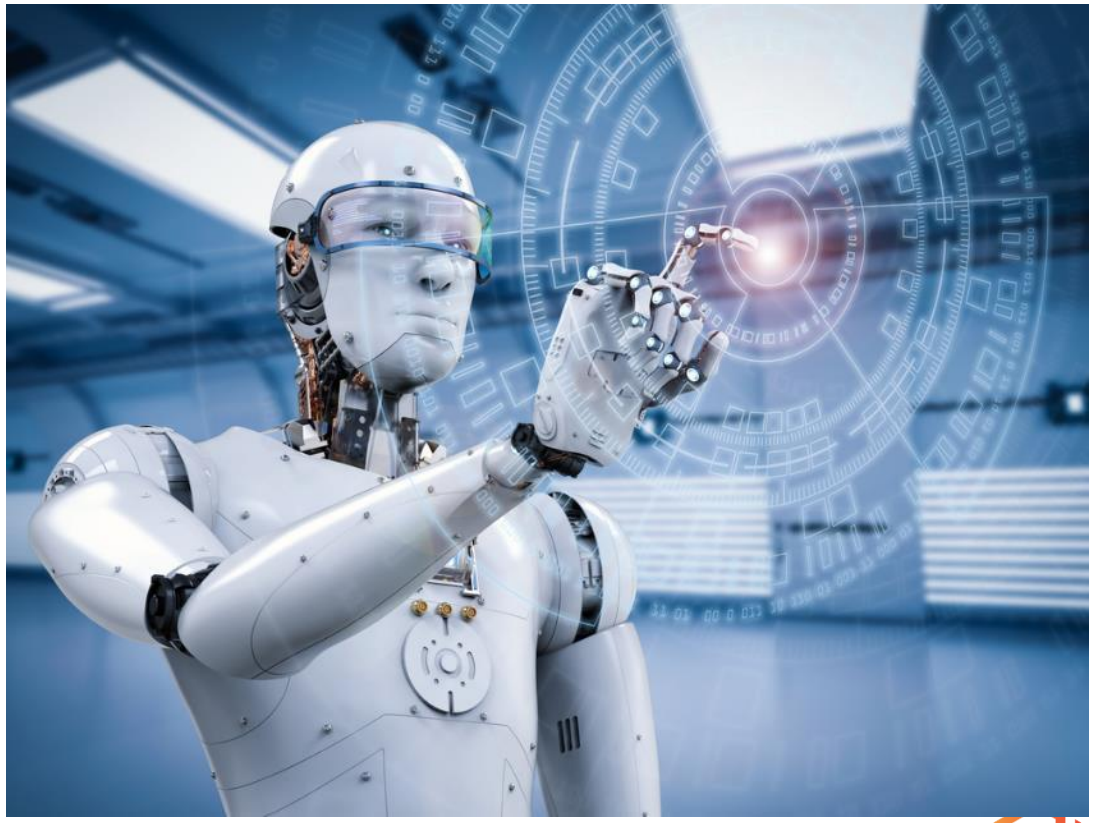


# Understanding AI

# AI Breakthrough Span 2 Categories

- Computer Vision

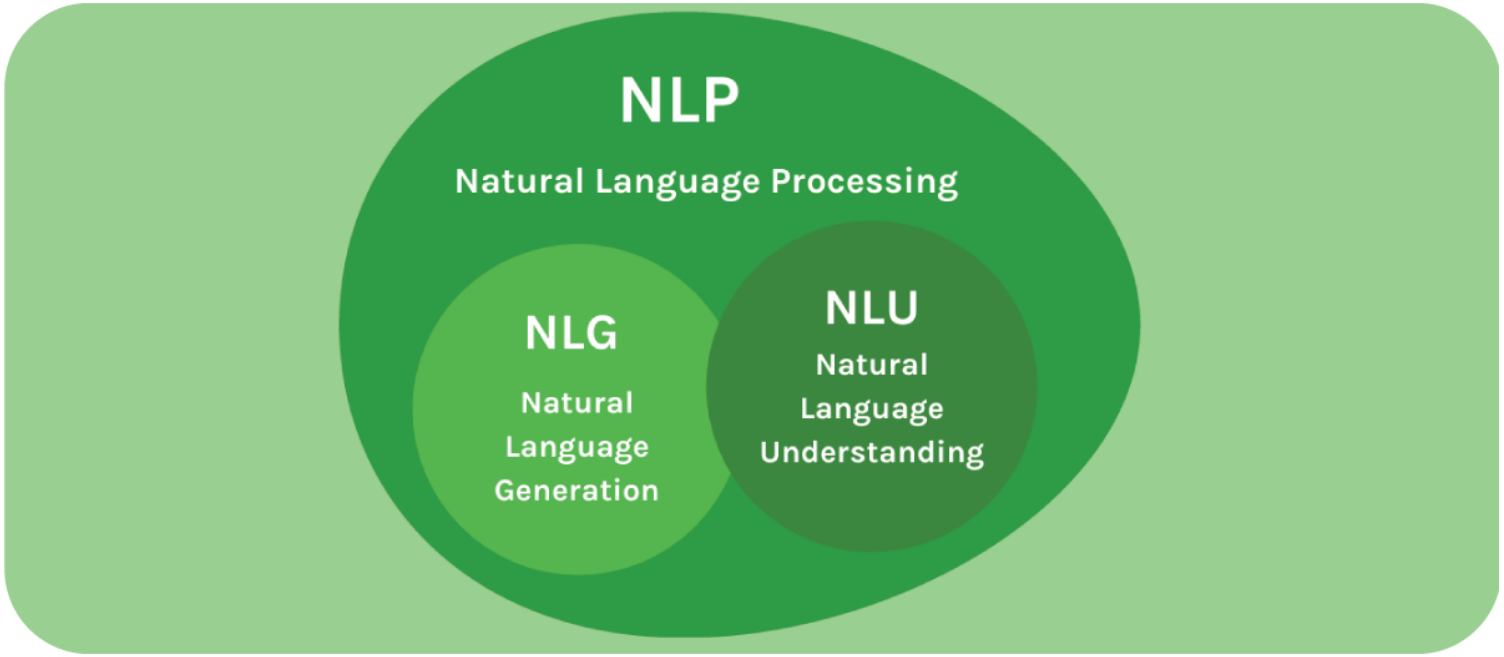

- Natural Language Processing



# What is NLP?

- **Wiki: Natural language processing** (**NLP**) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (**natural**) languages.

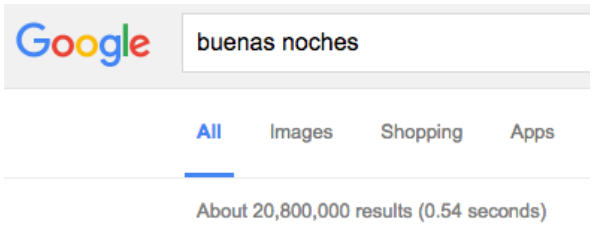
# What is NLP?


NLP
Natural Language Processing
NLG
Natural
Language
Generation
NLU
Natural
Language
Understanding

# What is NLP?

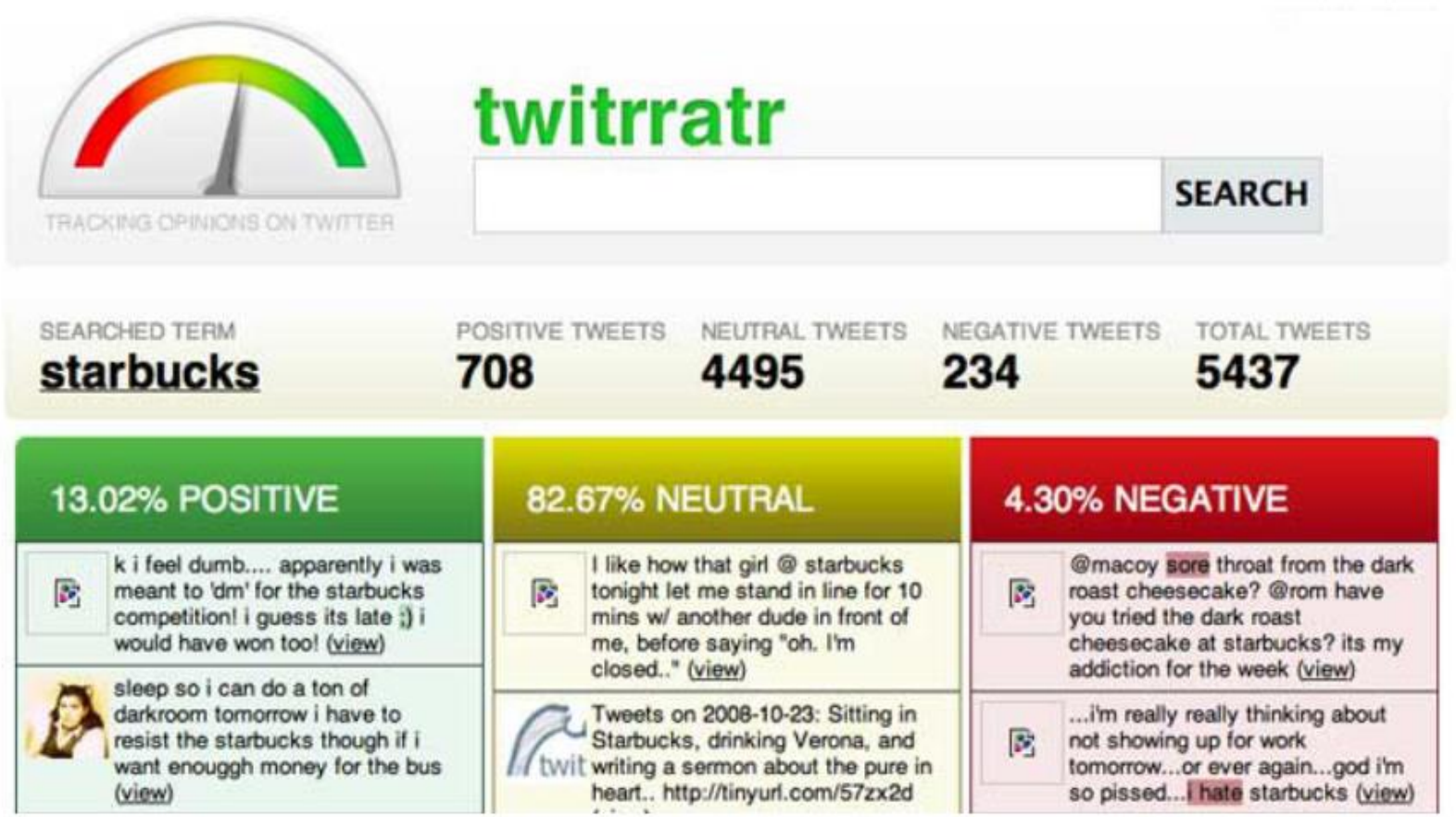
- “Natural” languages
  - English, Mandarin, French, Swahili, Arabic, Nahuatl, ….
  - NOT Java, C++, Perl, …
- Ultimate goal: Natural human-to-computer communication
- Sub-field of Artificial Intelligence, but very interdisciplinary
  - Computer science, human-computer interaction (HCI), linguistics, cognitive psychology, speech signal processing (EE), …
- MANY APPLICATIONS IN THE WORLD AROUND US

# Application: Machine Translation



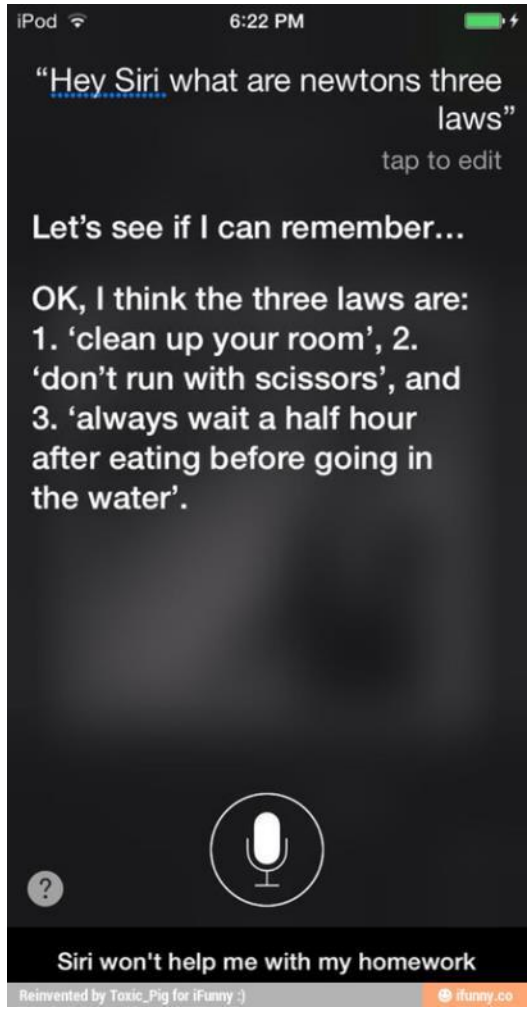



# Application: Sentiment Analysis



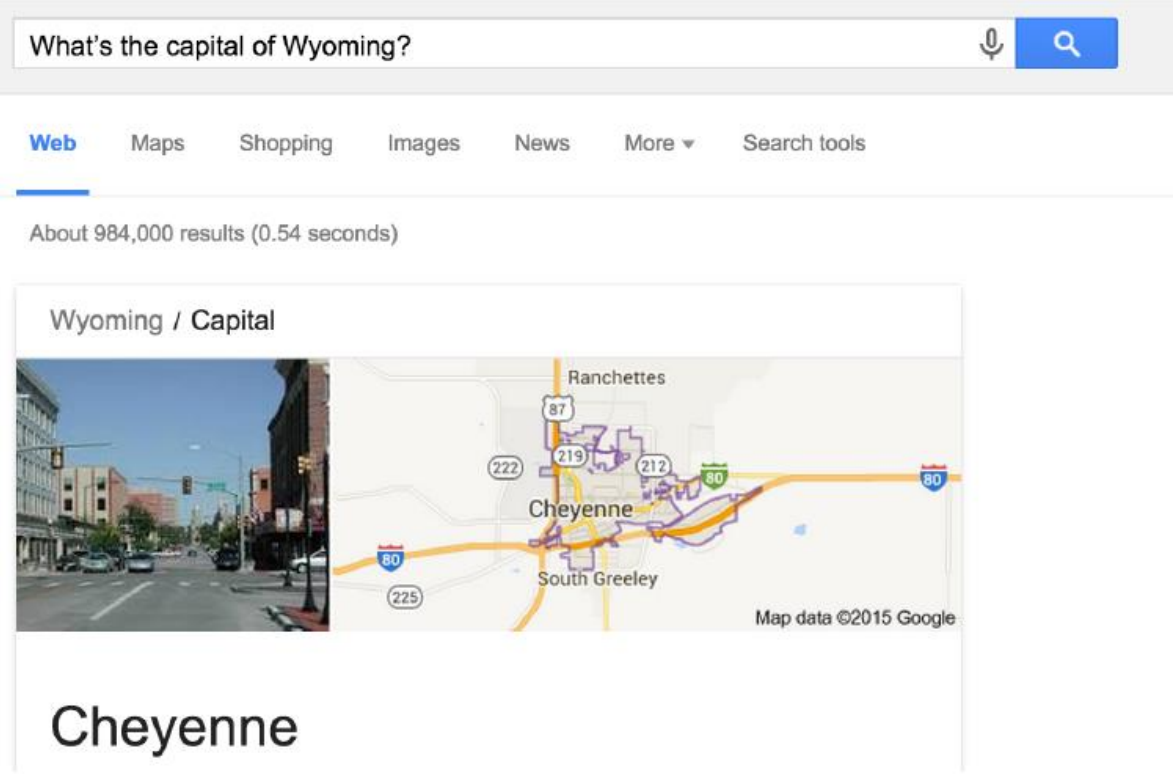
# Application: Text Classification

# Application: Question Answering


credit: ifunny.com

# It’s Everywhere

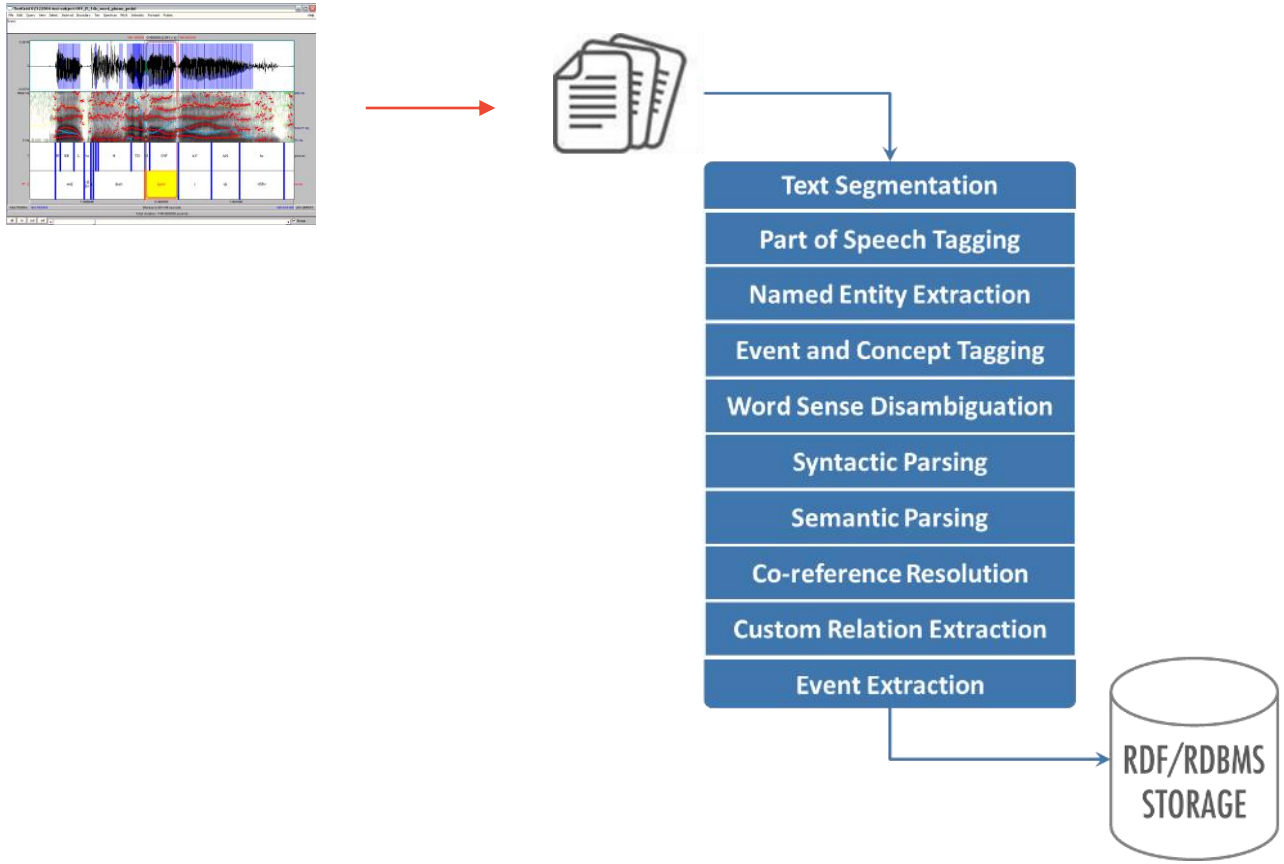



# Application: Information Extraction

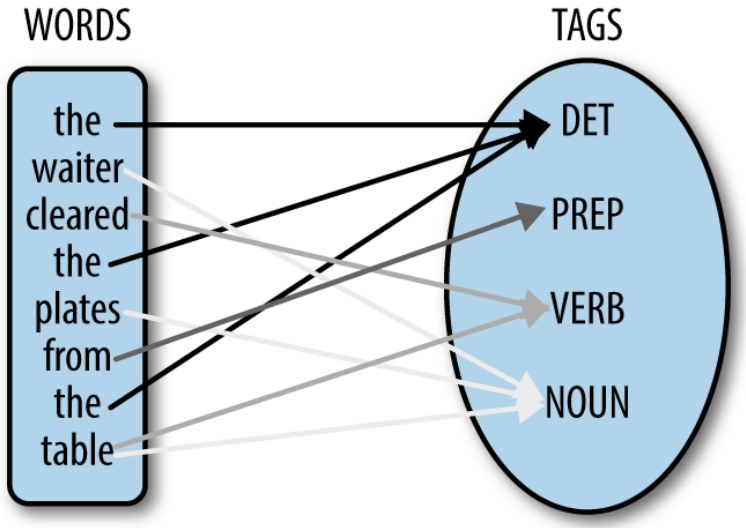
New York Times Co. named Russell T. Lewis, 45, president and general manager of its flagship New York Times newspaper, responsible for all business-side activities. He was executive vice president and deputy general manager. He succeeds Lance R. Primis, who in September was named president and chief operating officer of the parent.

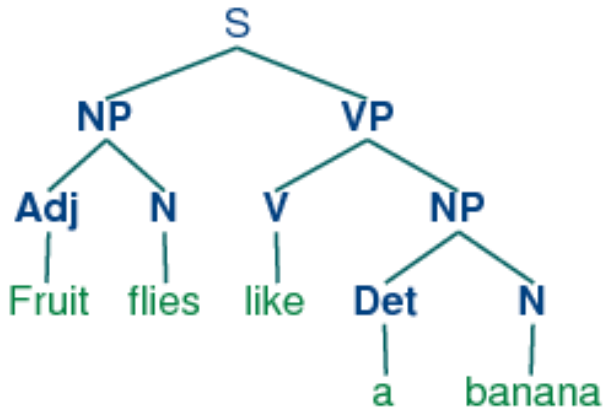
| Person | Company | Post | State |
|---|---|---|---|
| Russell T. Lewis | New York Times newspaper | president and general manager | start |
| Russell T. Lewis | New York Times newspaper | executive vice president | end |
| Lance R. Primis | New York Times Co. | president and CEO | start |

# Classic NLP Pipeline

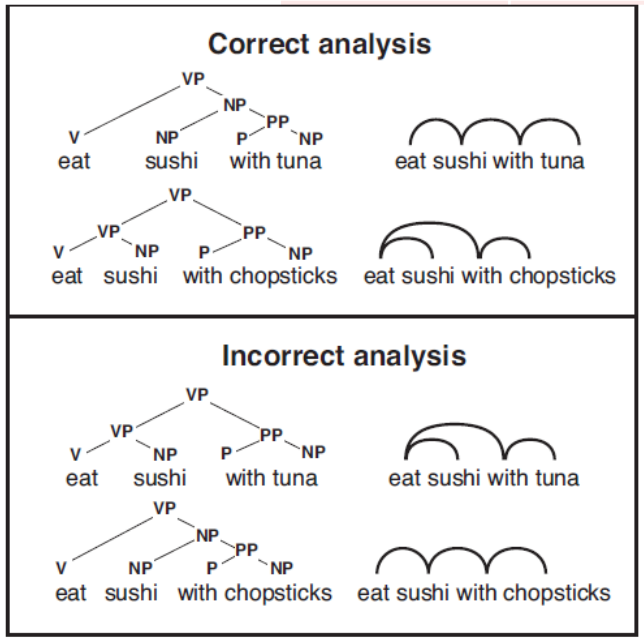


# Old School NLP (pre 2010)

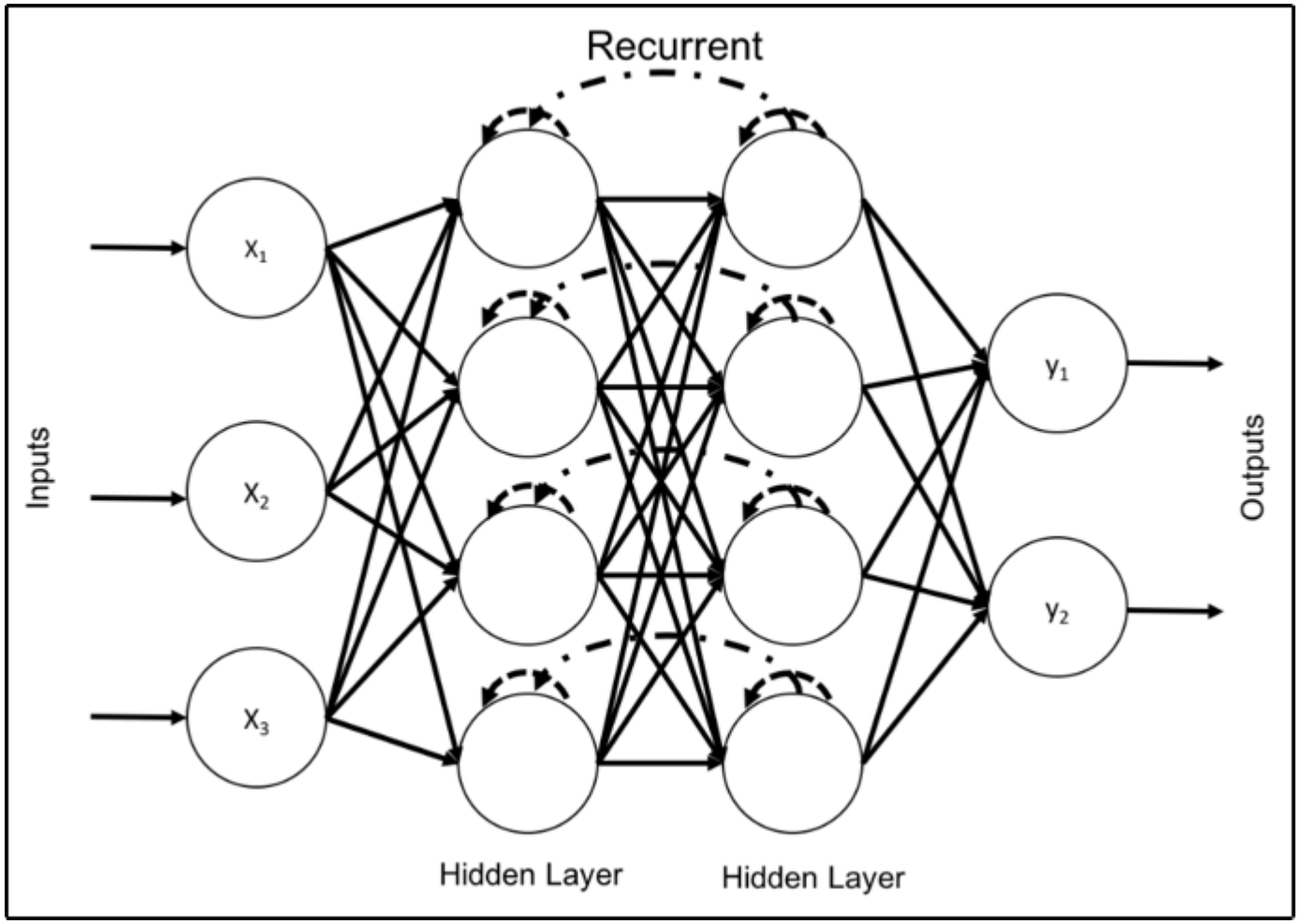


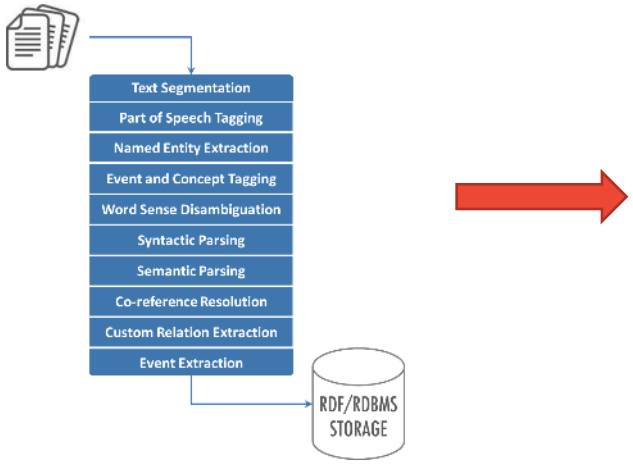

| Tag | Meaning | English Examples |
|---|---|---|
| ADJ | adjective | new, good, high, special, big, local |
| ADP | adposition | on, of, at, with, by, into, under |
| ADV | adverb | really, already, still, early, now |
| CONJ | conjunction | and, or, but, if, while, although |
| DET | determiner, article | the, a, some, most, every, no, which |
| NOUN | noun | year, home, costs, time, Africa |
| NUM | numeral | twenty-four, fourth, 1991, 14:24 |
| PRT | particle | at, on, out, over per, that, up, with |
| | | he, their, her, its, my, I, us |
| | | is, say, told, given, playing, would |
| n marks | | . , ; ! |
| | | ersatz, esprit, dunno, gr8, univeristy |

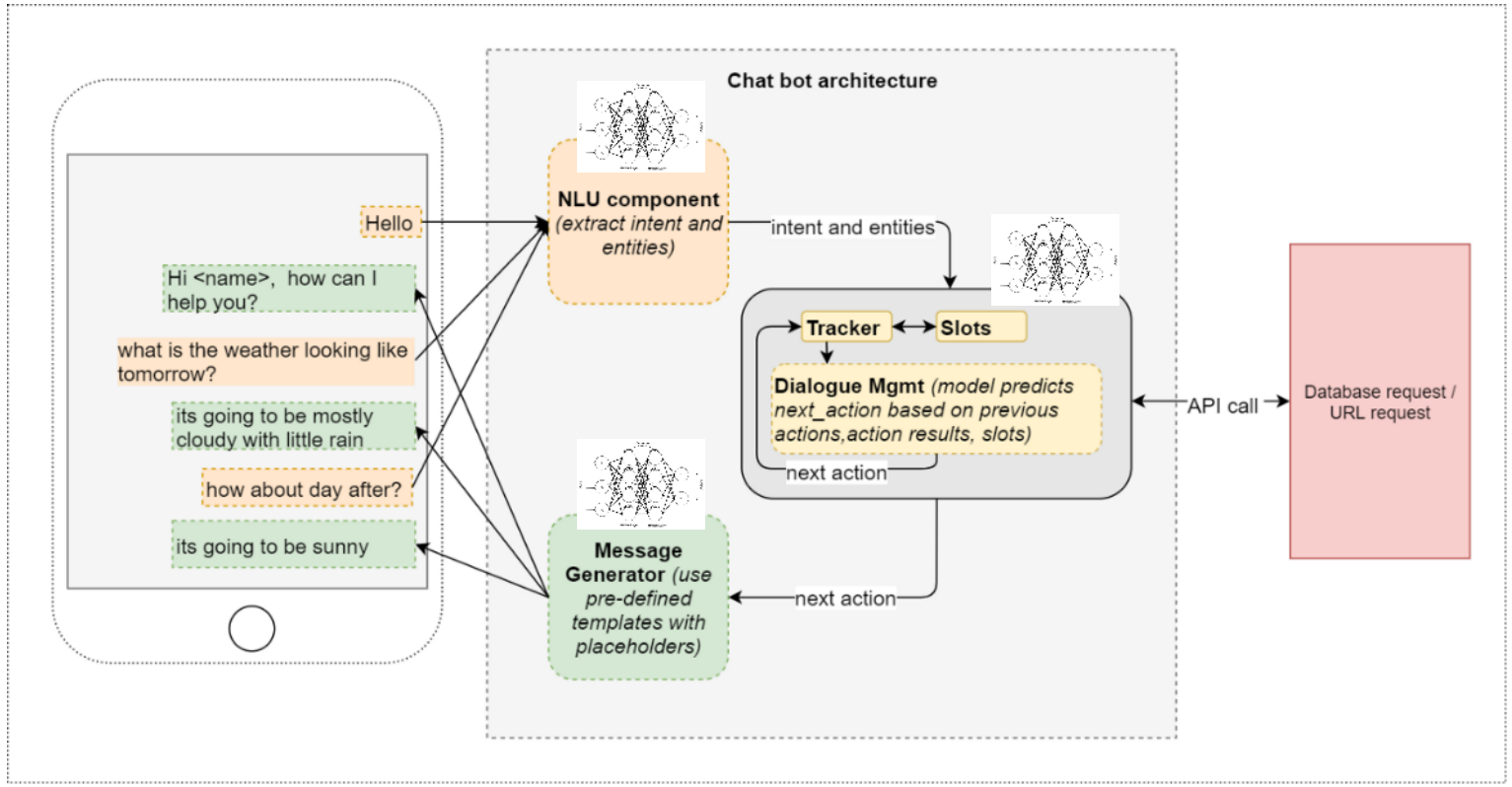


# Into the Future: Deep Learning, No Rules!

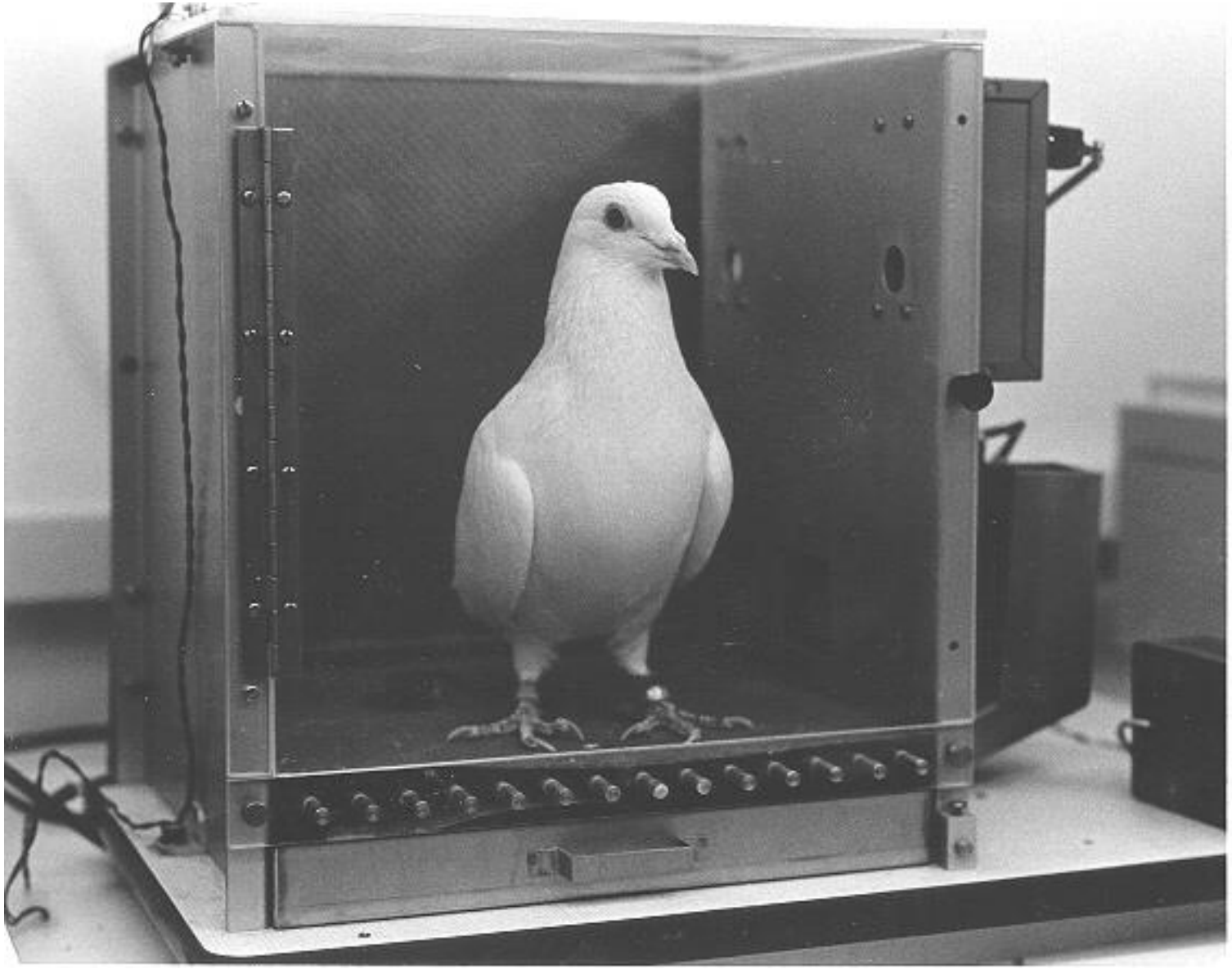


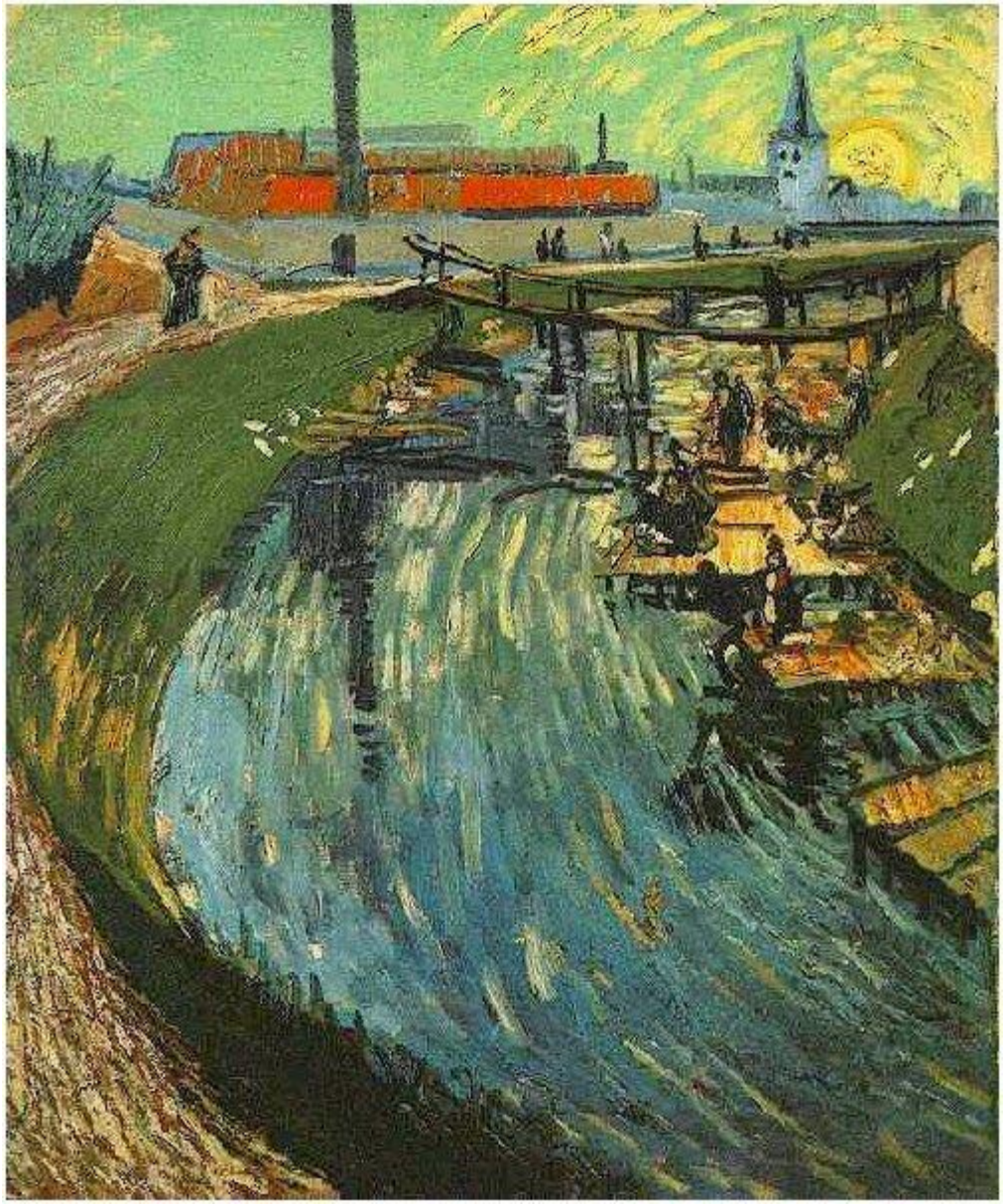
# Modern Conversational AI

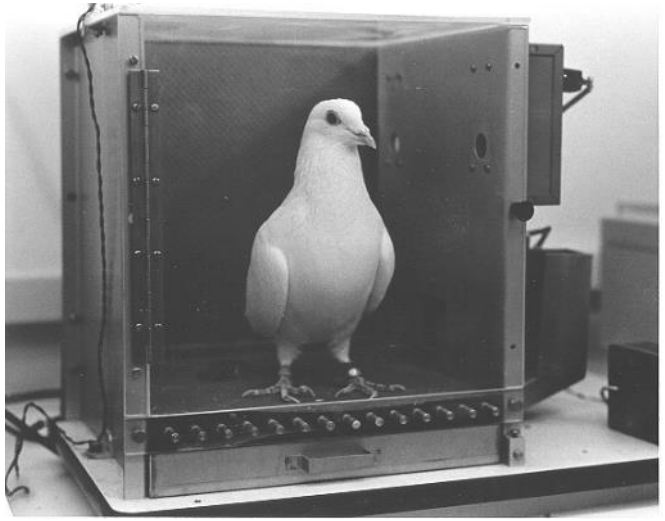


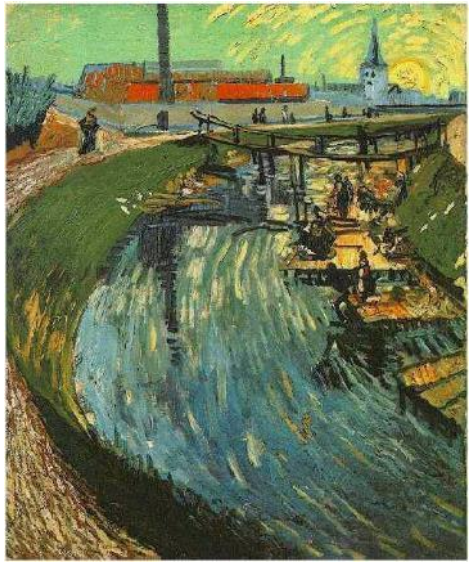



# Key AI Insight

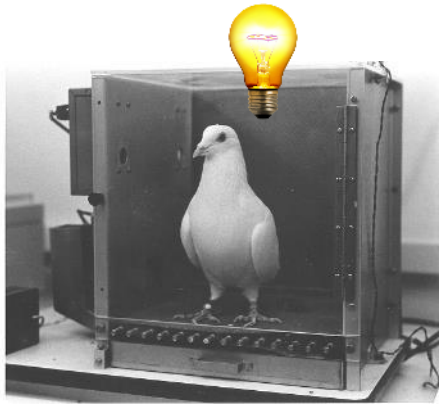






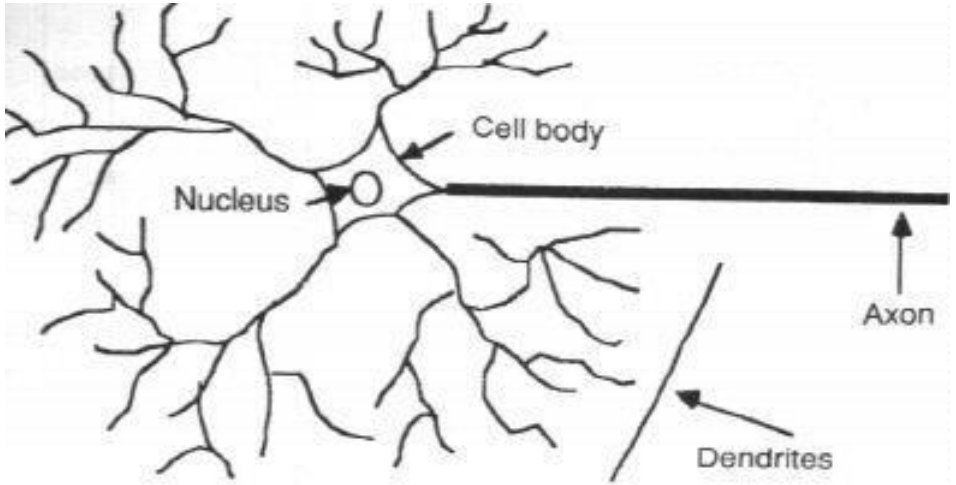


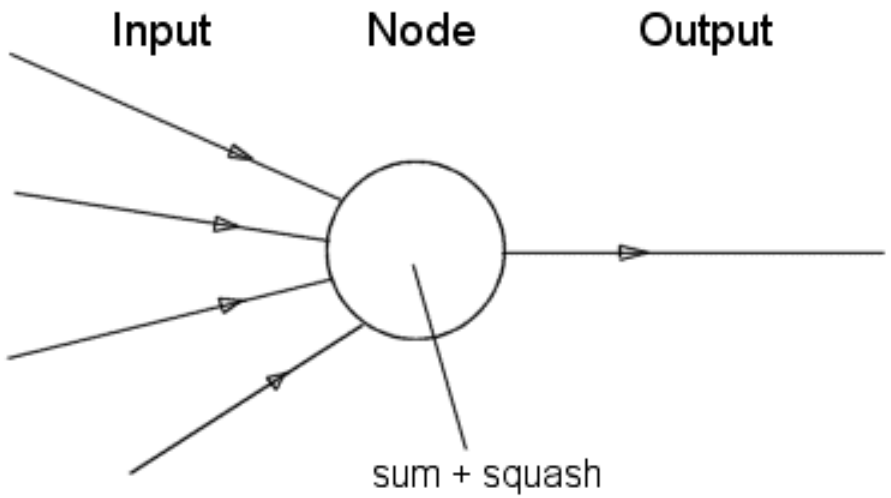
# Pigeon Brains are Strong!



- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists
- Pigeons do not simply memorise the pictures
- They can extract and recognise patterns (the ‘style’)
- They generalise from the already seen to make predictions
- This is what neural networks (biological and artificial) are good at (unlike conventional computer)

# Can We Model Brains in Computers? Yes!

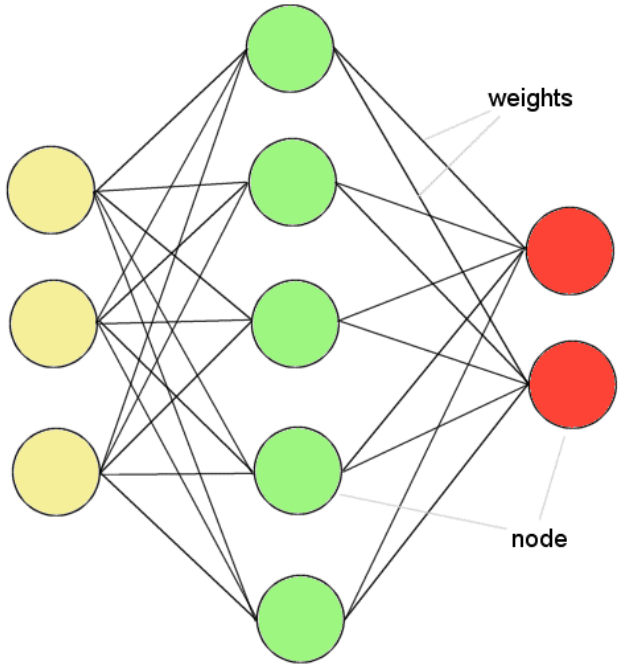


# In with the ANN

- **A**rtificial **N**eural **N**etworks incorporate the two fundamental components of biological neural nets:

1. Neurones (nodes)
2. Synapses (weights)



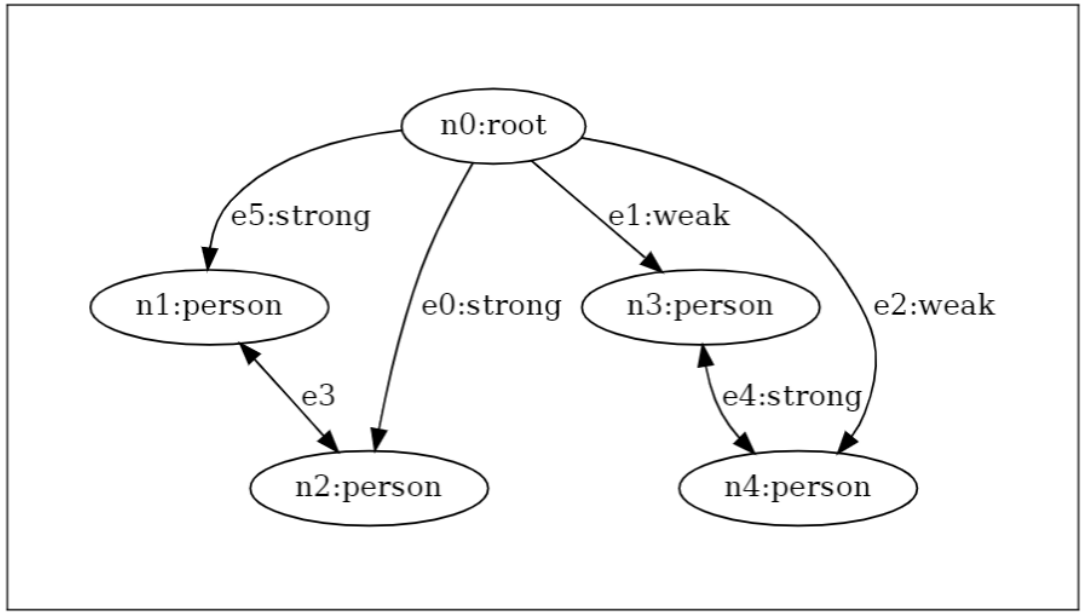
# Coding and Graphs

# Where We Left Off

- The Basics
  - Variables
  - Types
  - Loops
  - If statements
  - etc

```
walker init {
    fav_nums=[];

    for i=0 to i<10 by i+=1:
        fav_nums.l::append(i*2);
    std.out(fav_nums);

    fancy_str = "";
    for i in fav_nums {
        fancy_str = fancy_str + "two * " + i.str +
                    " = " + (i*2).str + ", ";
    }
    std.out(fancy_str);

    count_down = fav_nums[-1];
    while (count_down > 0) {
        count_down -= 1;
        if (count_down == 14):
            continue;
        std.out("I'm at countdown "+count_down.str);
        if (count_down == 10):
            break;
    }
}
```

# In Comes Graphs




- Spawn keyword used to spawn nodes and edges
- Can be assigned to variables, nodes are returned by default

```
node person {
    has name="Anon";
}

edge strong;
edge weak;

walker init {
        person1 = spawn here -[strong]->
node::person(name="Joe");
        person2 = spawn here -[strong]-> node::person;
        person3 = spawn here -[weak]-> node::person;
        person4 = spawn here -[weak]-> node::person(name="Mike");

        person1 <--> person2;
        person3 <-[strong]-> person4;

        for i in -->:
            std.out(i.context);
}
```

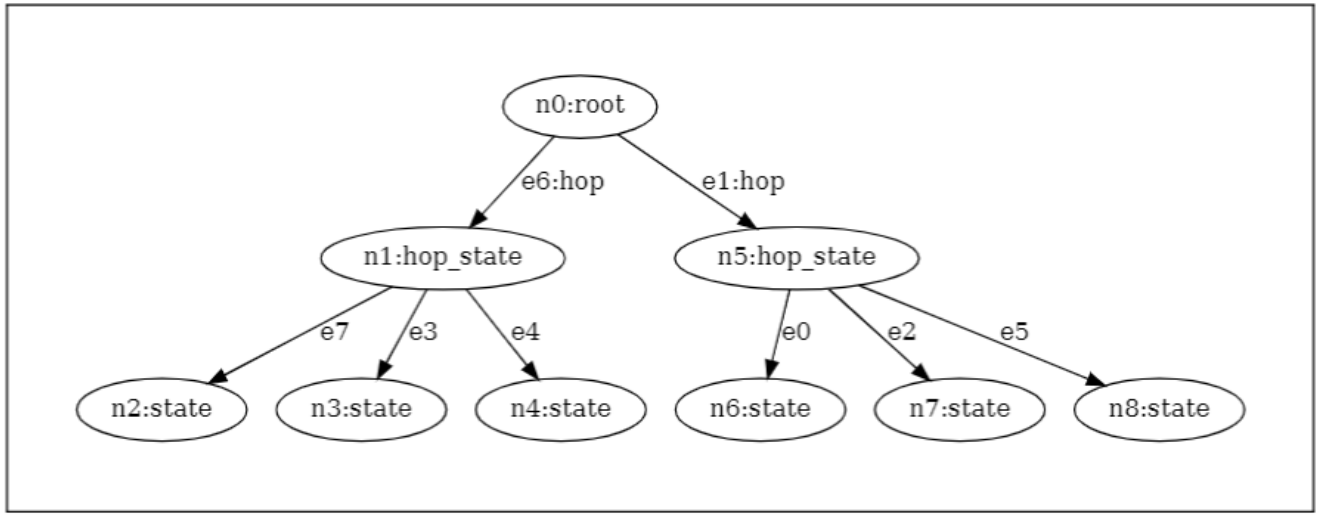
# Navigating Graphs

```
node state {
    has response="I'm silly state ";
}

node hop_state;

edge hop;

walker hop_buildout {
    spawn here --> node::state;
    spawn here --> node::state;
    spawn here --> node::state;
}

walker hop_counter {
    has anchor num=0; take -->; hop_state { num+=1; }
}
```

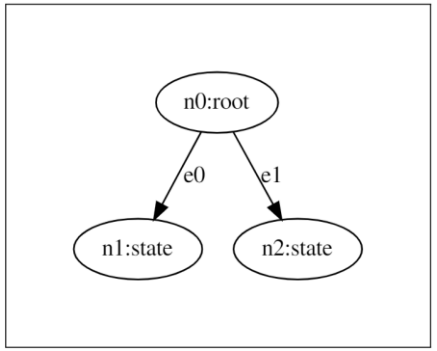
```
walker init {
    has state_visits=0, save_root;

    root {
        save_root = here;
        hop1 = spawn here -[hop]-> node::hop_state;
        hop2 = spawn here -[hop]-> node::hop_state;
    }

    hop_state:
        spawn here walker::hop_buildout;

    state {
        state_visits += 1;
        std.out(here.response+state_visits.str);
    }

    take -->;
    with exit {
        report spawn save_root walker::hop_counter;
    }
}
```

# Navigating Graphs

- Take command
  - Queue up connected node for walker to to go to next
  - Walker then executes full body on new node
  - Has variables are kept for journey
  - Node blocks will only execute on given node type



# Compute In Node

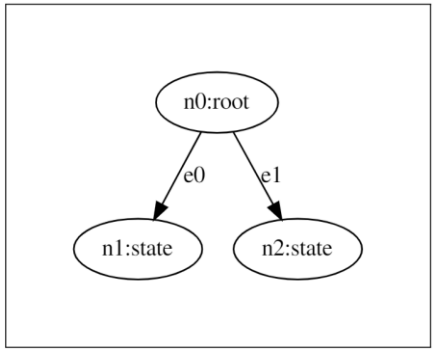



- Nodes can execute code when walkers arrive (or invoked)
- Visitor, like here, always points to the walkers context (scope)
- Execution can be triggered as walkers enter or exit nodes

```
node state {
    has name = rand.word().str::upper;
    has response = "I'm a silly bot. ";
    has user_utter;

    can speak with entry {
        std.out("I'm "+name+". And I currently have " +
visitor.info['name'] +
                " on me! ");
    }

    can listen with talker exit {
        user_utter = visitor.utterance;
        std.out("I heard '"+user_utter+"'\n");
        std.out(response);
    }

    can test_path with hop_counter entry {
        visitor.path.l::append(&here);
    }
}
```

# Compute In Node

- Walkers can be very simple
- Behavior specified in the nodes themselves



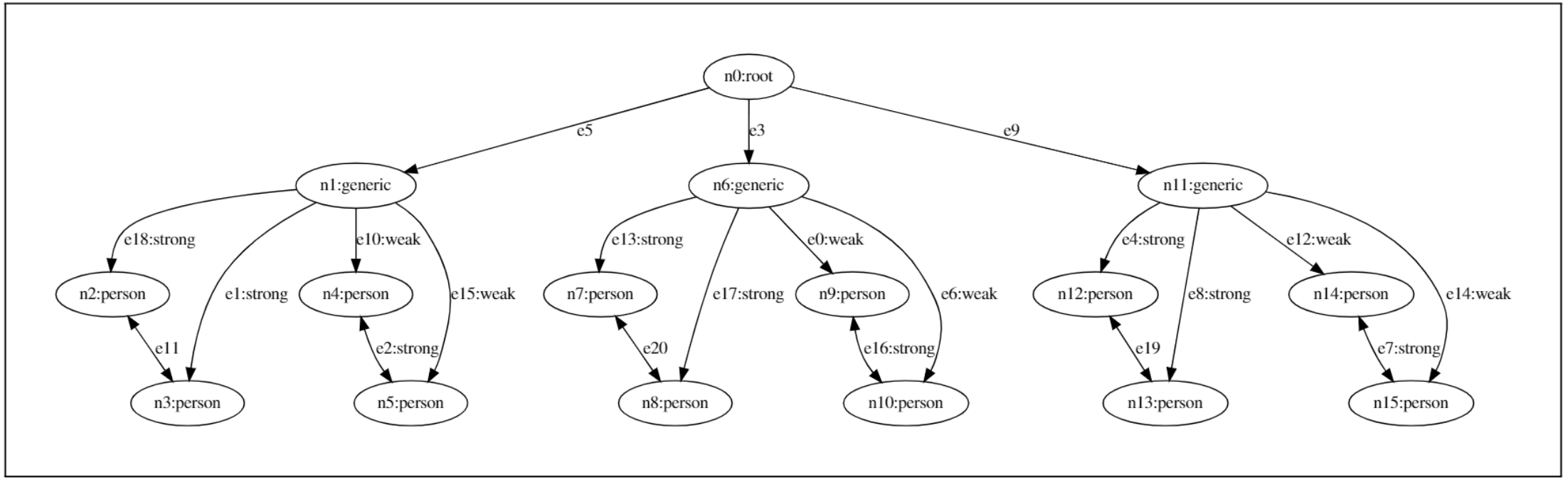

```
walker init {
    root {
        n1 = spawn here --> node::state;
        n2 = spawn here --> node::state;
    }
    spawn here walker::talker;
    spawn here walker::hop_counter;
}

walker talker {
    has utterance, path = [];
    utterance = rand.sentence();
    take -->;
}

walker hop_counter {
    has anchor path = [];
    take -->;

    with exit { std.out("\nHopper's path:", path); }
}
```

# Static Graphs

- Static graphs are ‘graph generators’
- Must specify an anchor node
- Handy for creating large graphs quicky
- Useful for tests!

```
node person {
    has name="Anon";
}

edge strong;
edge weak;

graph basic_gph {
    has anchor root;
    spawn {
        root = spawn node::generic;
        person1 = spawn root -[strong]-> node::person(name="Joe");
        person2 = spawn root -[strong]-> node::person;
        person3 = spawn root -[weak]-> node::person;
        person4 = spawn root -[weak]-> node::person(name="Mike");

        person1 <--> person2;
        person3 <-[strong]-> person4;
    }

}

walker init {
    spawn here --> graph::basic_gph;
    spawn here --> graph::basic_gph;
    spawn here --> graph::basic_gph;
}
```

# Static Graphs



# Tests

```
node person: has name="Anon";

graph basic {
    has anchor root;
    spawn {
        root = spawn node::generic;
        person1 = spawn root --> node::person(name="Joe");
        person2 = spawn root --> node::person;
        person3 = spawn root --> node::person;
        person4 = spawn root --> node::person(name="Mike");
        person1 <--> person2;
        person3 <--> person4;
    }

}

walker tally {
    has count=0, visited=[];
    count += 1;

    if(here not in visited) {
        visited.l::append(here);
        take -->;
    }
}
```

```
test "Size of basic graph"
with graph::basic by walker::tally {
    assert(visited.length == 5);
    assert(count > 5);
}

test "Size of a bit fancier graph"
with graph {
    has anchor root;
    spawn {
        root = spawn node::generic;
        spawn root --> graph::basic; spawn root -->
graph::basic;
    }
} by walker::tally {
    assert(visited.length == 11);
    assert(count > 11);
}
```

# Tests

- Tests are very powerful for developing code
- Can specify a graph for a walk to walk on and assert values make sense

```
ninja@DESKTOP-VO9IVBR:~/jac_convAI/tinker$ jsctl -m jac test tldr_test.jac
Testing "Size of basic graph": [PASSED in 0.00s]
Testing "Size of a bit fancier graph": [PASSED in 0.01s]
{
  "tests": 2,
  "passed": 2,
  "failed": 0,
  "success": true
}
ninja@DESKTOP-VO9IVBR:~/jac_convAI/tinker$
```

```
test "Size of basic graph"
with graph::basic by walker::tally {
    assert(visited.length == 5);
    assert(count > 5);
}

test "Size of a bit fancier graph"
with graph {
    has anchor root;
    spawn {
        root = spawn node::generic;
        spawn root --> graph::basic; spawn root -->
graph::basic;
    }
} by walker::tally {
    assert(visited.length == 11);
    assert(count > 11);
}
```