Principles of Software Construction: Objects, Design, and Concurrency

Events Everywhere!

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University School of Computer Science

institute for SOFTWARE RESEARCH

Administrivia

Reminder: HW5

A comment on midsemester grades

- (including participation)

There was a reading quiz today! It's on Canvas, or:

There is also a reading for Thursday!

Outline

- Revisiting Core vs. GUI
- Model-View-Controller
- Client server programming, and TicTacToe
- ReactJS UI
- Event-Based Programming, Reactive Programming

Core vs GUI

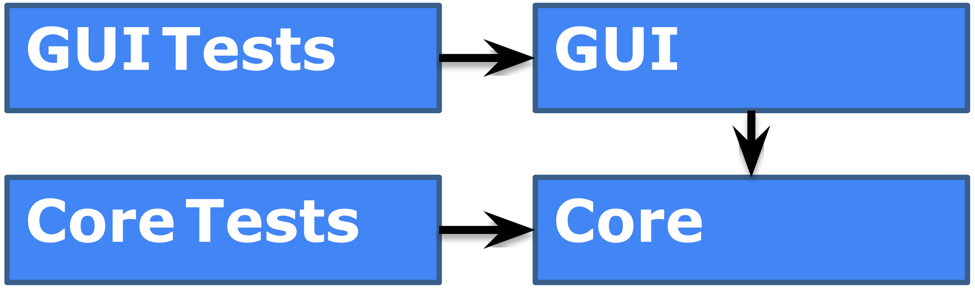
Backend vs Frontend

Recall: Core implementation vs. GUI

- Core implementation: application logic
  - Computing some result, updating data
- GUI
  - Graphical representation of data
  - Source of user interactions
- Design guideline: *avoid coupling the GUI with core application*
  - Multiple UIs with single core implementation
  - Test core without UI

Recall: Separating application core and GUI

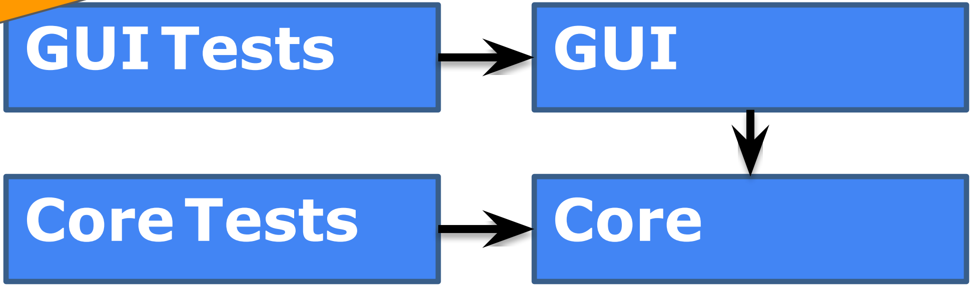
- Reduce coupling: do not allow core to depend on UI
- Create and test the core without a GUI
  - Use the Observer pattern to communicate information from the core (Model) to the GUI (View)



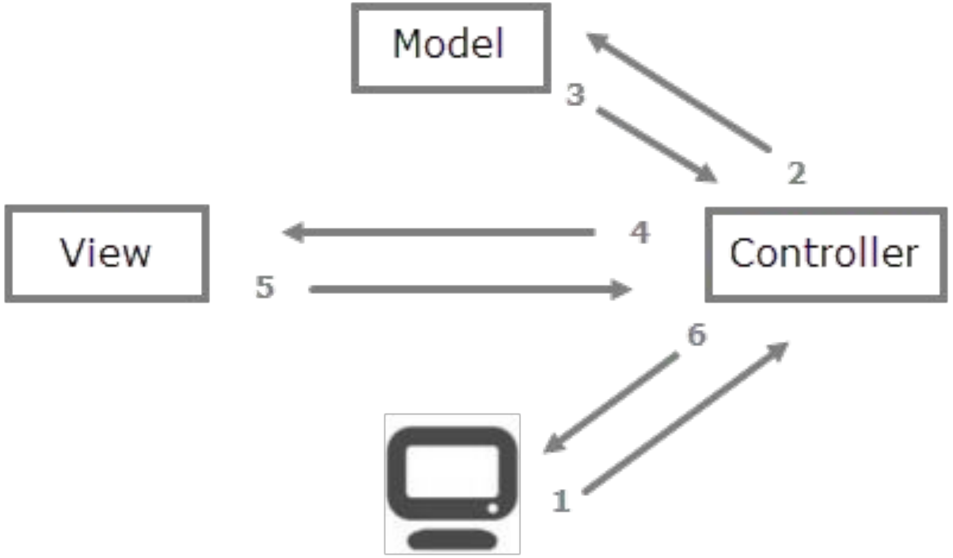
Recall: Separating application core and GUI

- Reduce coupling: do not allow core to depend on UI
- Create and test the core without a GUI
  - Use the Observer pattern to … om the core (Model) to the G…

What design goals does this further?

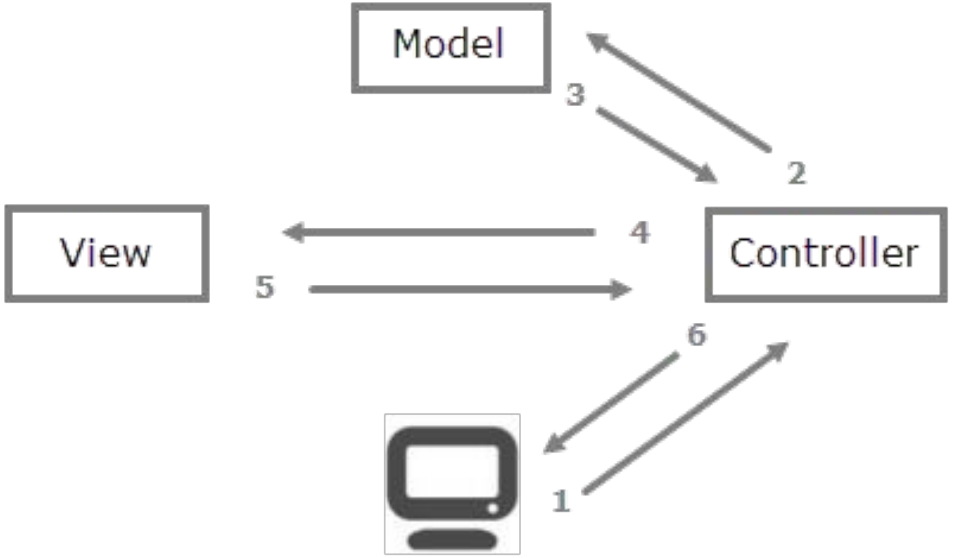


Model View Controller



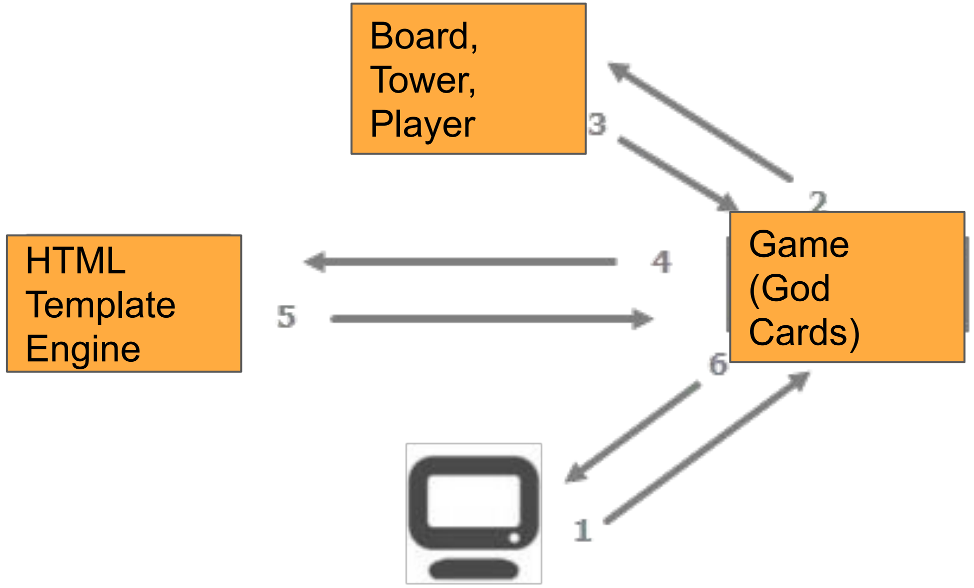
https://overiq.com/django-1-10/mvc-pattern-and-django/

Model View Controller in Santorini?



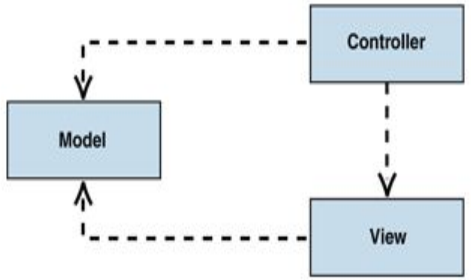
https://overiq.com/django-1-10/mvc-pattern-and-django/

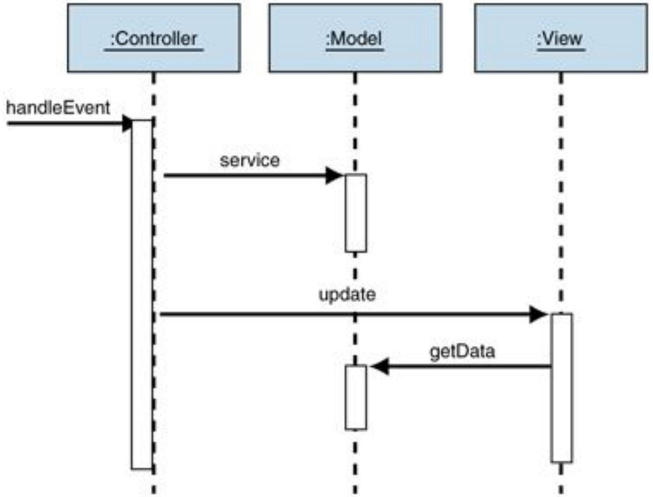
Model View Controller in Santorini?



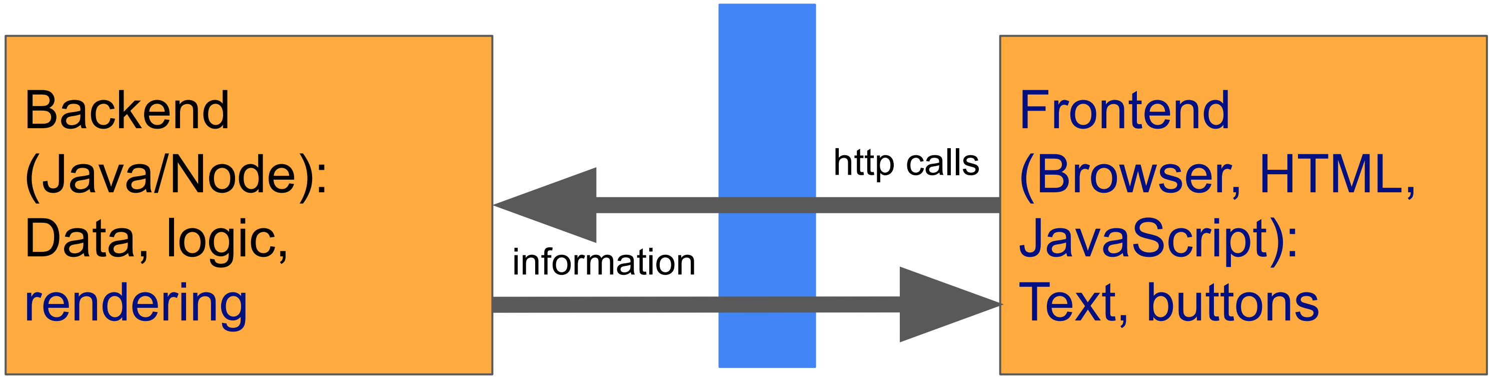
https://overiq.com/django-1-10/mvc-pattern-and-django/

Model View Controller Dependencies



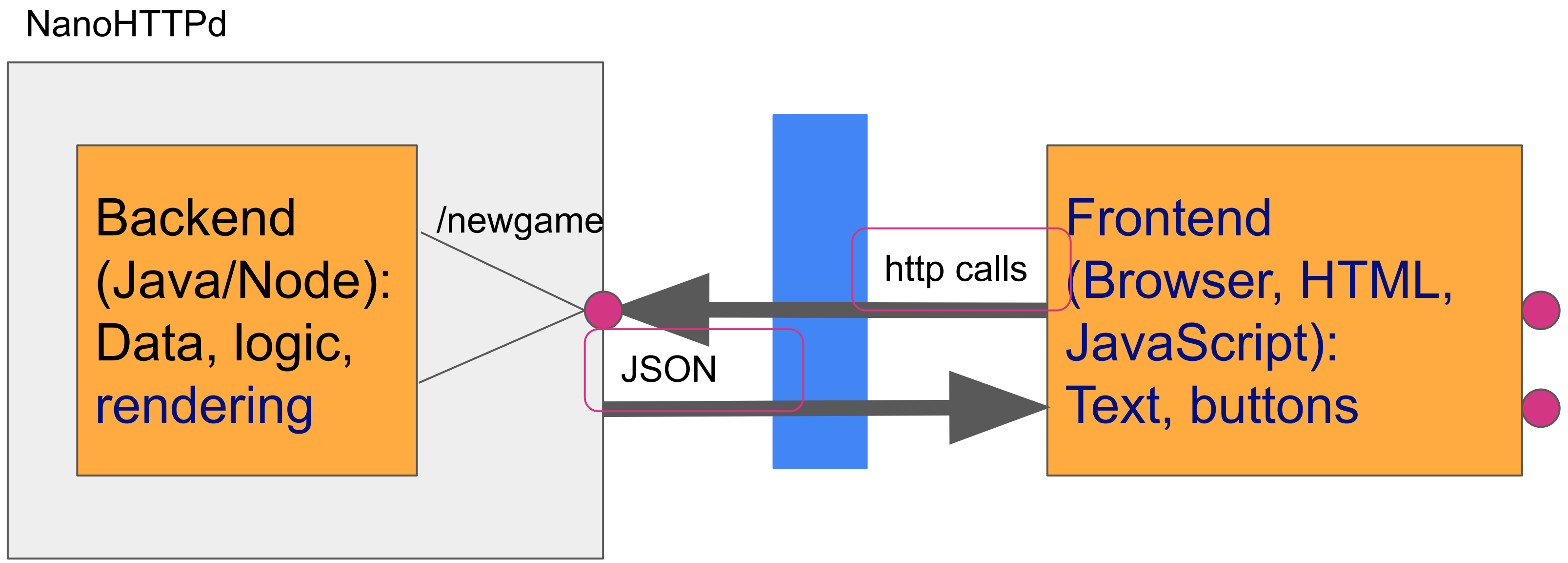


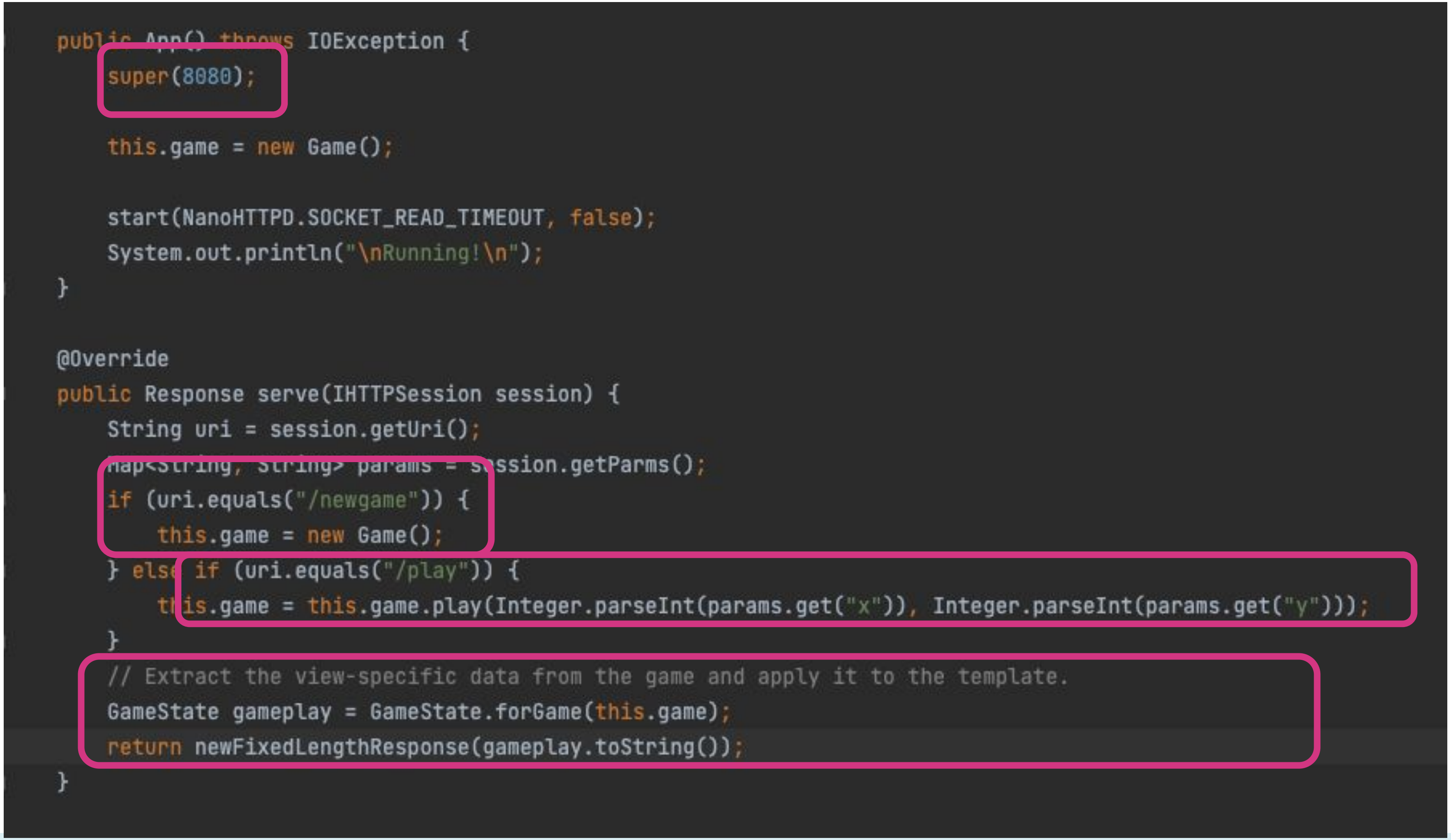
Client-Server Programming forces Frontend-Backend Separation



Browser can call web server, but not the other way around
Browser needs to *pull* for updates
Browser can request entire page, or just additional content (ajax, REST api calls, …)

TicTacToe


NanoHTTPd
Backend (Java/Node): Data, logic, rendering
/newgame
http calls
JSON
Frontend (Browser, HTML, JavaScript): Text, buttons



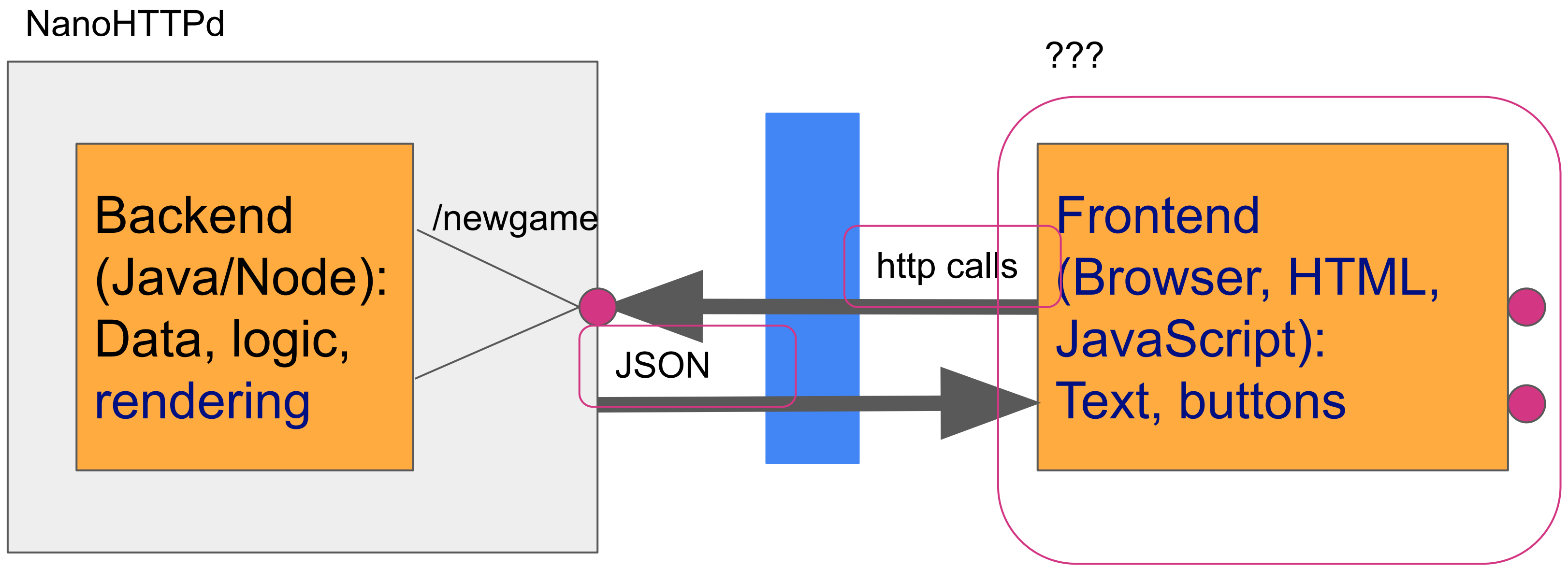
```java
public App() throws IOException {
    super(8080);

    this.game = new Game();

    start(NanoHTTPD.SOCKET_READ_TIMEOUT, false);
    System.out.println("\nRunning!\n");
}

@Override
public Response serve(IHTTPSession session) {
    String uri = session.getUri();
    Map<String, String> params = session.getParms();
    if (uri.equals("/newgame")) {
        this.game = new Game();
    } else if (uri.equals("/play")) {
        this.game = this.game.play(Integer.parseInt(params.get("x")), Integer.parseInt(params.get("y")));
    }
    // Extract the view-specific data from the game and apply it to the template.
    GameState gameplay = GameState.forGame(this.game);
    return newFixedLengthResponse(gameplay.toString());
}
```

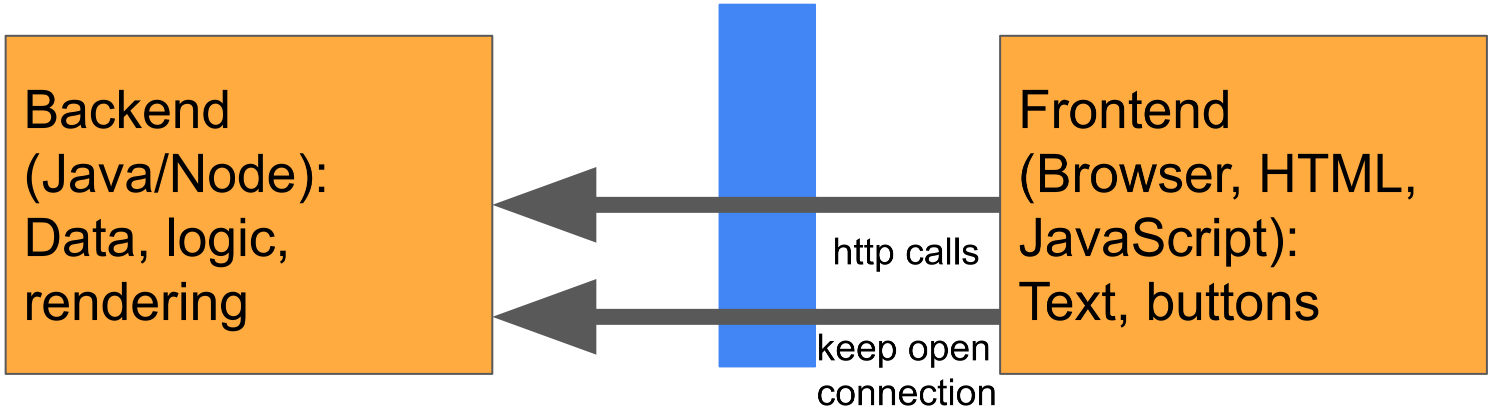
TicTacToe



NanoHTTPd
Backend (Java/Node): Data, logic, rendering
/newgame
JSON
http calls
???
Frontend (Browser, HTML, JavaScript): Text, buttons

Some alternatives

Client-Server Programming forces Frontend-Backend Separation


Backend (Java/Node): Data, logic, rendering
http calls
keep open connection
Frontend (Browser, HTML, JavaScript): Text, buttons

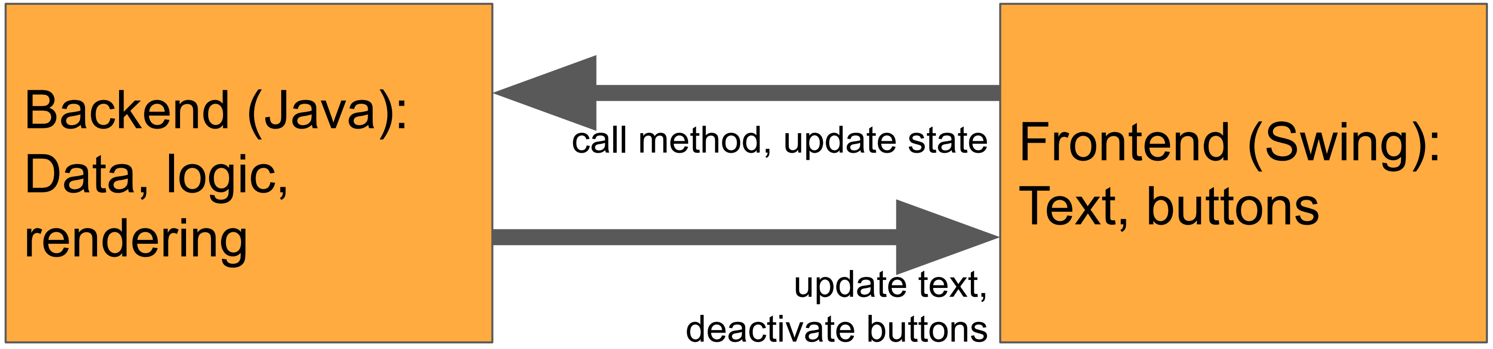
Trick to let backend push information to frontend: Keep http request open, append to page (compare to stream)
Alternative: regular pulling

Core & Gui in same environment

JavaScript frontend and backend together in browser
(e.g. using *browserify*) -- single threaded!

Java Swing GUI running in same VM as core logic -- multi threaded

Core logic could directly modify GUI

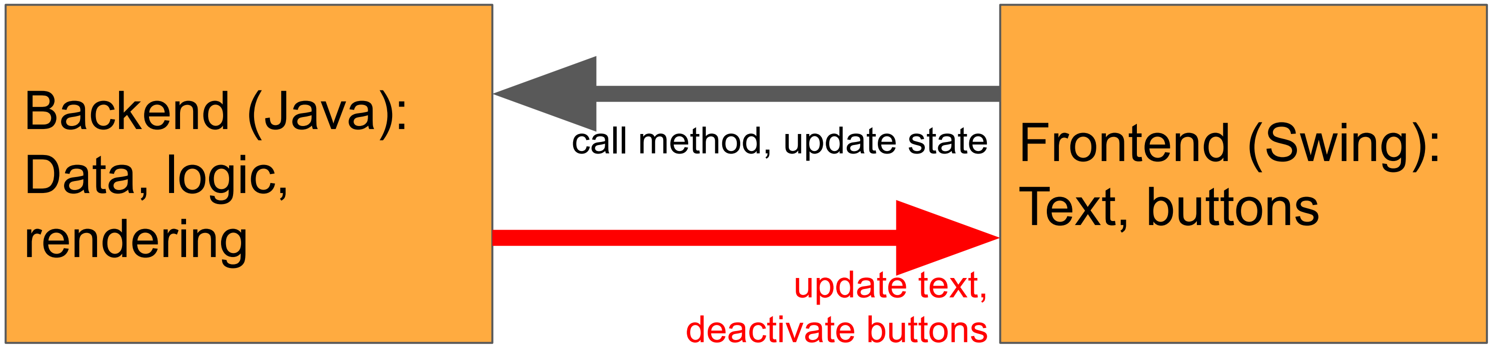


Backend (Java): Data, logic, rendering
call method, update state
update text, deactivate buttons
Frontend (Swing): Text, buttons

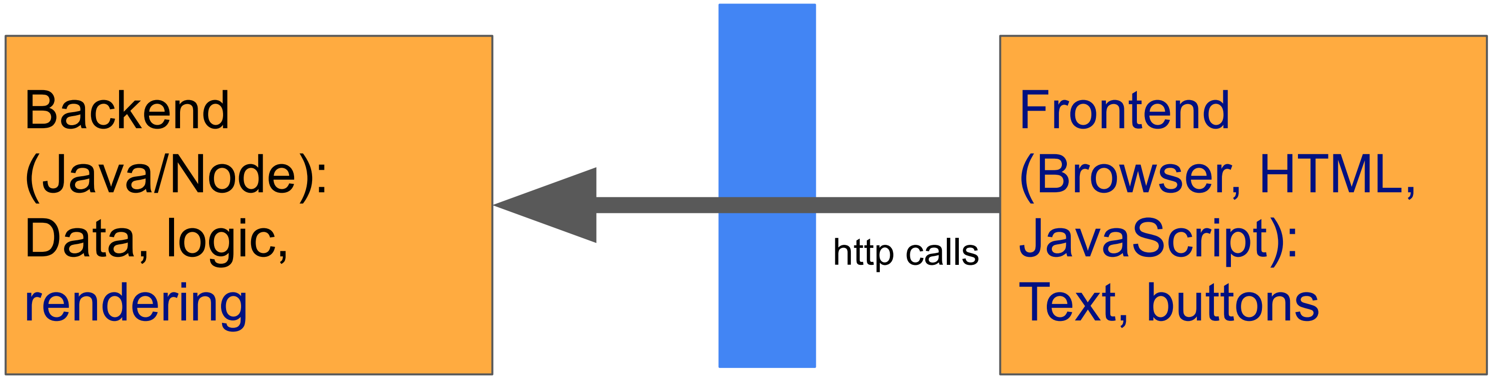
Avoid Core to Gui coupling NOTE: WE FORCE YOU TO NOT DO THIS

Never call the GUI from the Core

Update GUI after action (pull) or use observer pattern instead to inform GUI of updates (push)


Backend (Java): Data, logic, rendering
call method, update state
Frontend (Swing): Text, buttons
update text, deactivate buttons

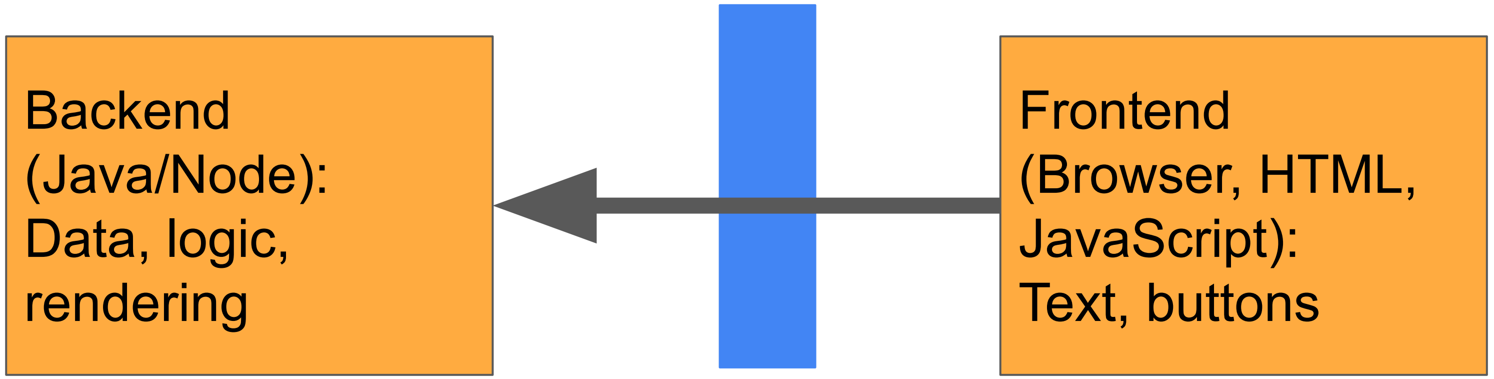
GUI Code in the Backend


Backend (Java/Node): Data, logic, rendering
http calls
Frontend (Browser, HTML, JavaScript): Text, buttons

Typically there is some GUI code in Backend (rendering/view)
Could also send entire program state to frontend (e.g, json) and render there with JavaScript

Where to put GUI Logic?

Example: Deactivate undo button in first round of TicTacToe, deactivate game buttons after game won

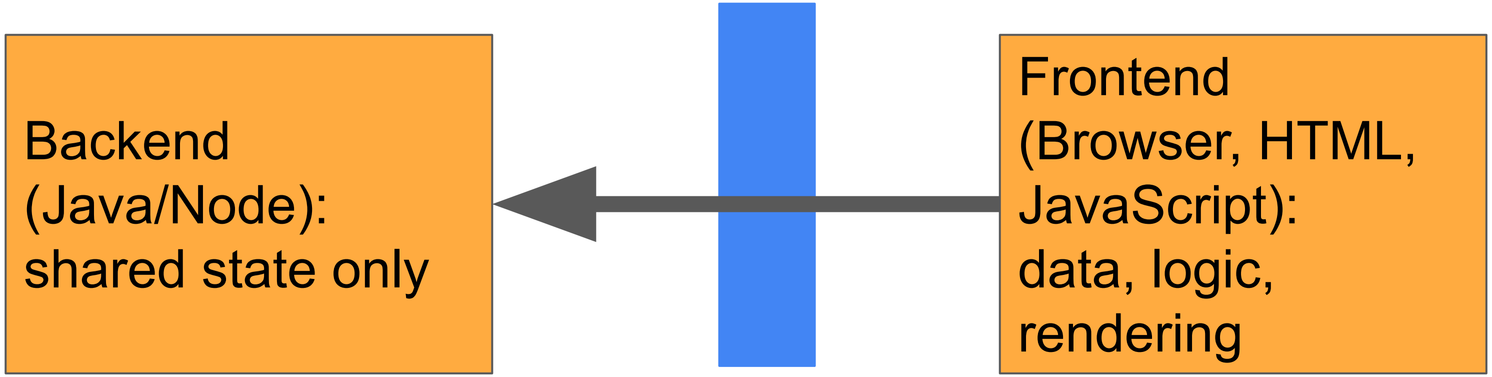

Backend (Java/Node): Data, logic, rendering
Frontend (Browser, HTML, JavaScript): Text, buttons

Option 1: All rendering in backend, update/refresh the entire page after every action -- simpler
Option 2: Handle some logic in frontend, use backend for checking -- fewer calls, more responsive

Core Logic in Frontend?

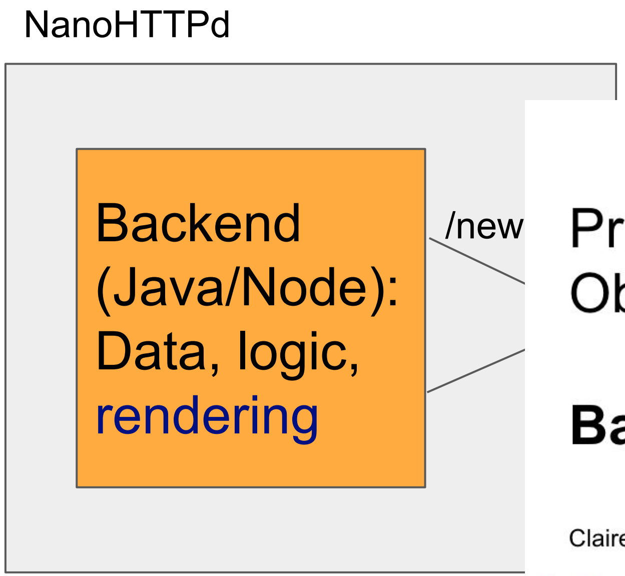
Could move core logic largely to client, minimize backend interaction

Can frontend be trusted? Need to replicate core in front and backend?

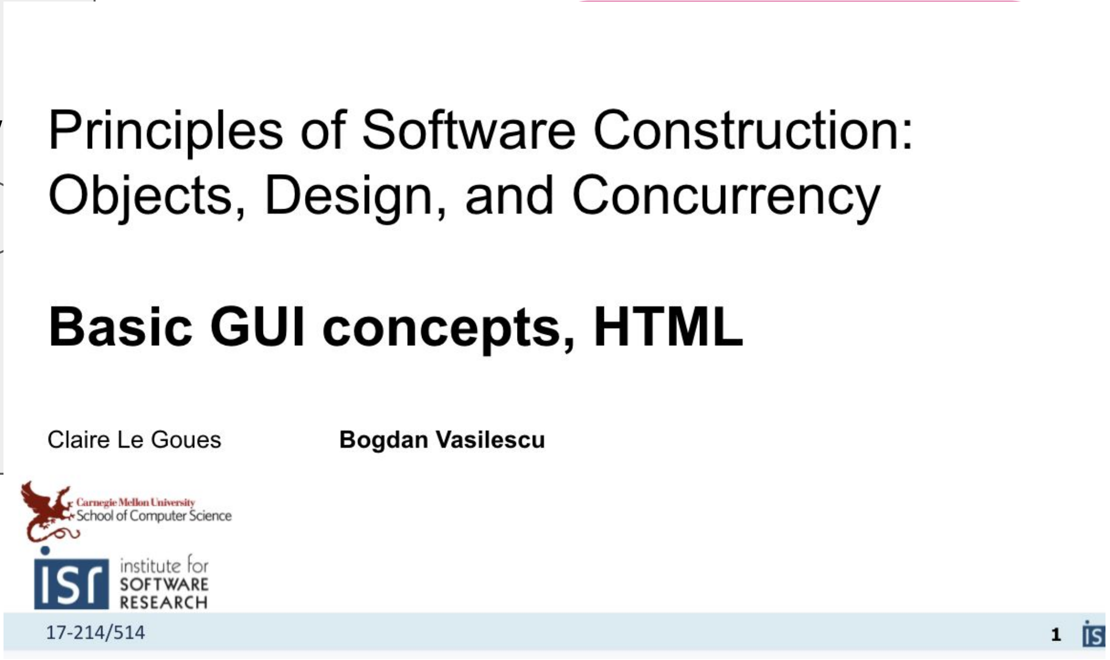

Backend (Java/Node): shared state only
Frontend (Browser, HTML, JavaScript): data, logic, rendering

(React and other frameworks make it easy to introduce logic in the frontend; avoid tangling all core logic with GUI)

TicTacToe



???


Principles of Software Construction: Objects, Design, and Concurrency

Basic GUI concepts, HTML

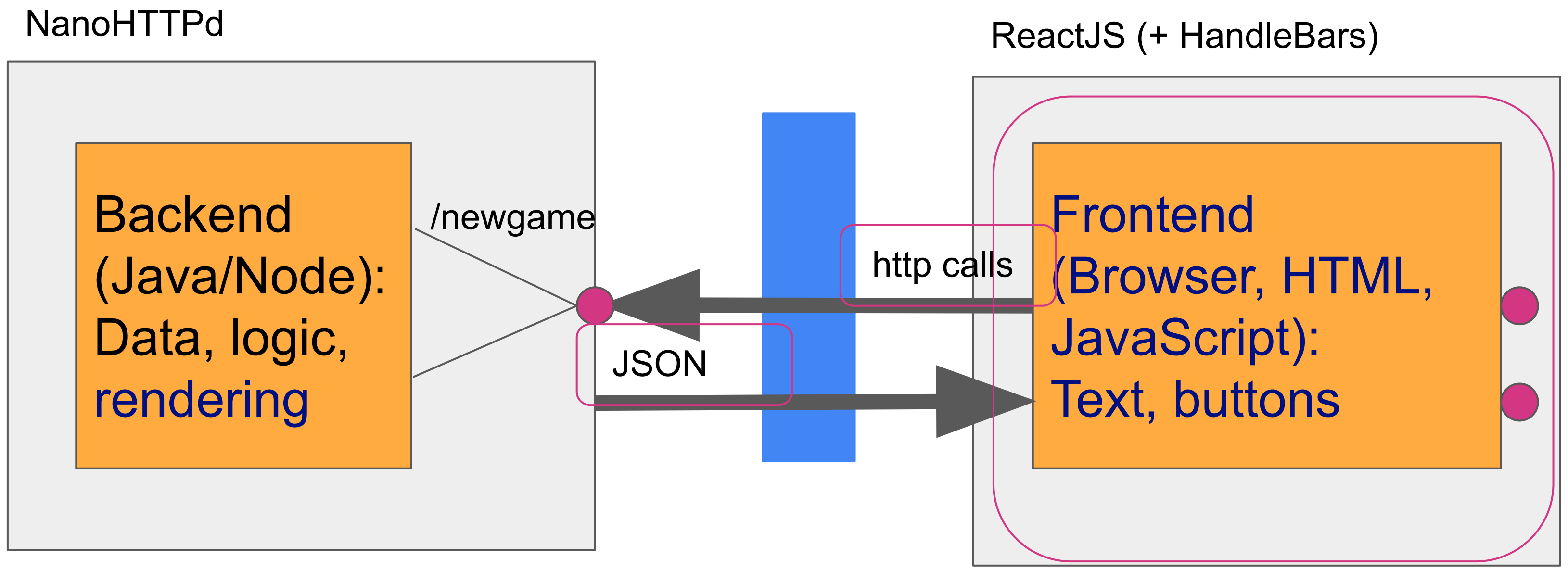
Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University School of Computer Science

institute for SOFTWARE RESEARCH

17-214/514 1

TicTacToe


NanoHTTPd
Backend (Java/Node): Data, logic, rendering
/newgame
http calls
JSON
ReactJS (+ HandleBars)
Frontend (Browser, HTML, JavaScript): Text, buttons

ReactJS

ReactJS

Popular frontend library by Facebook

Template library and state management

(Not a reactive programming library, though it adopts some similar ideas – we'll get back to reactive programming)

Templates with ReactJS

(Similar ideas to Handlebars in HW4 and Rec7)

Describe rendering of HTML, inputs given as objects

JSX language extension to embed HTML in JS

Try it:
https://reactjs.org/redirect-to-codepen/introducing-jsx

```
function formatName(user) {
  return user.firstName + ' ' +
         user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>Hello, {formatName(user)}!</h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Composing Templates

(Corresponds to Fragments in Handlebars)

Nest templates

Pass arguments (properties) between templates

Try it: https://reactjs.org/redirect-to-codepen/components-and-props/composing-components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
function App() { return (
    <div>
     <Welcome name="Sara" />
     <Welcome name="Edite" />
   </div>
  );}
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Templates with State

Class notation instead of function

If state changes, page is re-rendered

Try it:
https://codepen.io/gaearon/pen/xEmzGg?editors=0010

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
    this.handleClick = this.handleClick.bind(this);
 }
 handleClick() {
   this.setState(prevState => ({
     isToggleOn: !prevState.isToggleOn
   }));
 }
 render() { return (
     <button onClick={this.handleClick}>
       {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );  }
}

ReactDOM.render(
  <Toggle />,
  document.getElementById('root')
);
```

ReactJS Templates

Can use arbitrary JavaScript code (Handlebars can only access object properties)

Properties are read-only

State is mutable and *observed* for re-rendering (state updates are asynchronous)

Re-rendering is optimized and asynchronous, will rerender inner components too if their properties change

ReactJS and Core Logic

React makes it easy to add functionality in GUI

This really tangles GUI and logic (violating separation argued for previously)

Suggestion: Use React state primarily for UI-related logic (e.g., selecting workers) and keep the core logic in the backend or as a separate library -- be very explicit about what information is shared

Connecting React to Some Core

Use observer pattern to let react component observe changes

Encapsulate in *useEffect()* hook

Further discussion: https://reactjs.org/docs/hooks-custom.html

```
function App() {
  const [data, setData] =
            React.useState(null);
  React.useEffect(() => {
    function handleStatChange(e) {
      setData(e.updatedData);
    }
    CoreAPI.subscribe(handleStatChange);
    return () => {
      CoreAPI.unsubscribe(handleStatChange);
    };
  });
  return (
    <div>/* using state in data */</div>
  );
```

Connecting React to backend

Return json from server backend and store as component state

Full example: https://www.freecodecamp.org/news/how-to-create-a-react-app-with-a-node-backend-the-complete-guide/

```
function App() {
  const [data, setData] =
              React.useState(null);
  React.useEffect(() => {
    fetch("/api")
      .then((res) => res.json())
      .then((data) =>
         setData(data.message));
  }, []);

  return (
    <div>/* using state in data */</div>
  );
}
```

React and Homework 5/6

Using React is entirely optional

We showed you how to use Handlebars + React in Rec07

Many other template engines and frontend frameworks exists (e.g., Vue, Angular, …)

React adds complexity but also easy updates reacting to state changes

Reactive Programming

Reactive Programming

Programming strategy or patterns, where programs react to data

Embraces concurrency, focuses on data flows

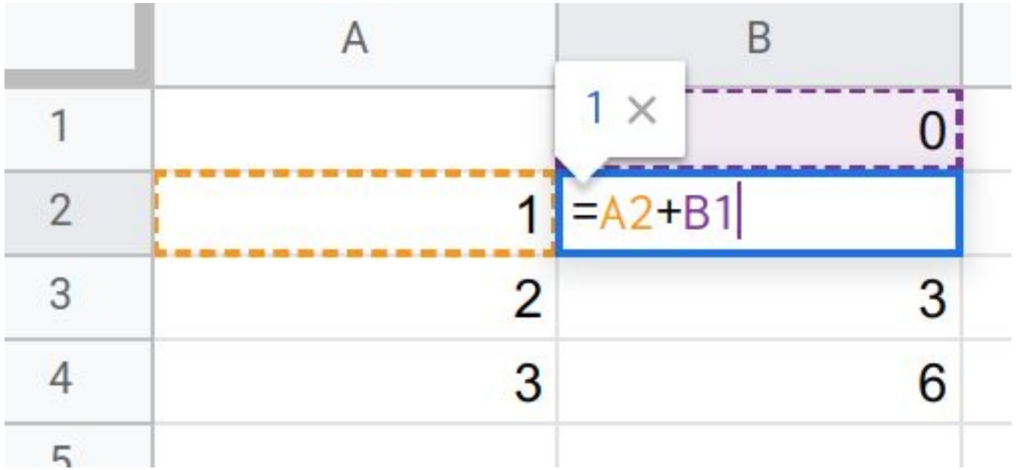
Takes event-based programming to an extreme

Decouples programs around data

Useful analogy: Spreadsheets

Cells contain data or formulas

Formula cells are computed automatically whenever input data changes



	A	B
1		0
2	1	=A2+B1
3	2	3
4	3	6
5		

Implementing Spreadsheet-Like Computations?

Implementing Spreadsheet-Like Computations?

```
x = 3

y = 5

z = x + y

print(z) // prints 8

x = 5

print(z) // expect 10, prints 8
```

in imperative computations,
no update when inputs change

Implementing Spreadsheet-Like Computations?

```
x = 3

y = 5

z = () => x + y

print(z()) // prints 8

x = 5

print(z()) // prints 10
```

computation performed on demand (pull)
caching possible

Does not easily work in Java, since Java requires variables in closure to be final. Need object with mutable internal state

Implementing Spreadsheet-Like Computations?

```
x = new Cell(3)

y = new Cell(5)

z = new DerivedCell(x, y, (a,b)=>a+b)

print(z.get()) // prints 8

x.set(5)

print(z.get()) // prints 10
```

Cell implements observer pattern,
informs observers of changes (push)

DerivedCell listens to changes from Cell,
updates internal state on changes,
informs own observers of changes

Complications

Single change in cell can trigger many computations (push)

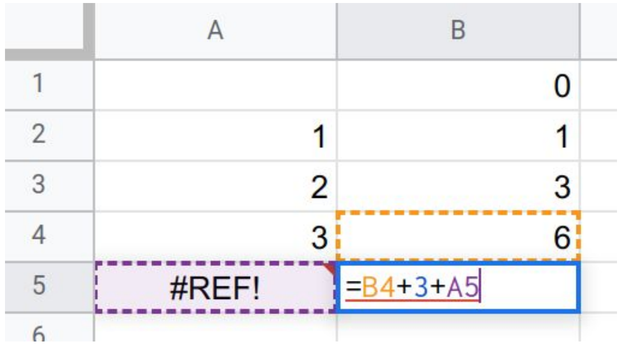
Possibly put in queue, compute asynchronously

Perform some computations lazily when needed

Cyclic dependencies can result in infinite loops

Detect, special ways to handle

Observers can hinder garbage collection



	A	B
1		0
2	1	1
3	2	3
4	3	6
5	#REF!	=B4+3+A5
6		

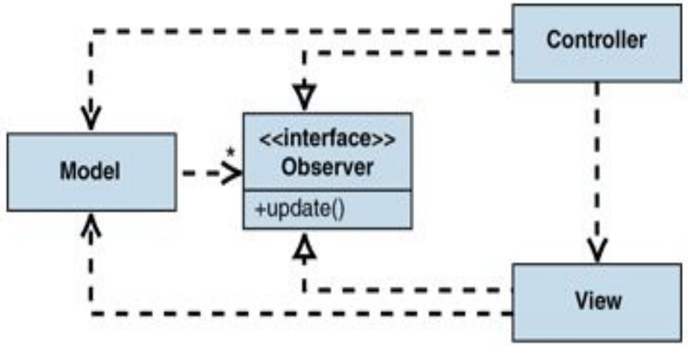
Reactive Programming and GUIs

Store state in observable cells, possibly derived

Have GUI update automatically on state changes

Have buttons perform state changes on cells

Mirrors active model-view-controller pattern, discussed later (model is observable cell)



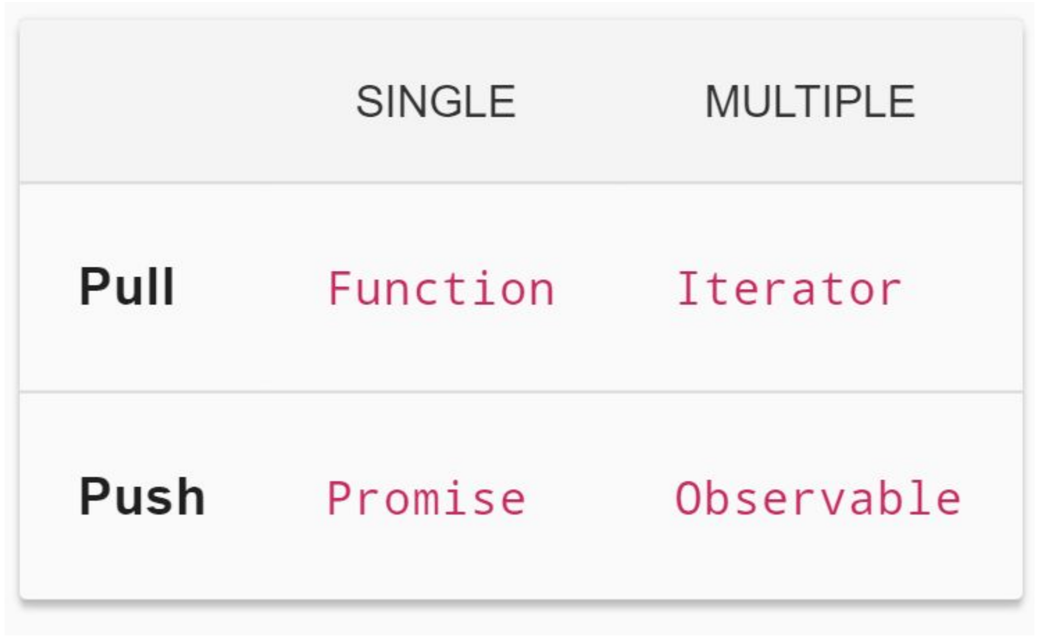
From Pull to Push

Instead of clients to look for state (pull)

observers react to state changes with actions (push)

Commonly observables indicate that something has changed, triggering observers to get updated state (push-pull)

Beyond Spreadsheet Cells



	SINGLE	MULTIPLE
Pull	Function	Iterator
Push	Promise	Observable

https://rxjs.dev/guide/observable

Reactive Programming Libraries

RxJava, RxJS, many others

Provide Stream-like interfaces for event handling, with many convenience functions (similar to promises)

Observables typically allow pushing multiple values in sequence

Cells can be implemented by considering only the latest value of observables

Previous Example with RxJava

```
PublishSubject<Integer> x = PublishSubject.create();
PublishSubject<Integer> y = PublishSubject.create();
Observable<Integer> z = Observable.combineLatest(x, y,
(a,b)->a+b);
z.subscribe(System.out::println);
x.onNext(3);
y.onNext(5);
x.onNext(5);
```

Chaining Computations along Data

```
awk '{print $7}' < /var/log/nginx/access.log |
    sort |
    uniq -c |
    sort -r -n |
    head -n 5 > out
```

Multiple programs executed in sequence each read lines and produce lines;
can start reading lines before previous program is finished

Streams / Reactive Programming / Events

Instead of calling methods in sequence,
set up pipelines for data processing

Let data control the execution

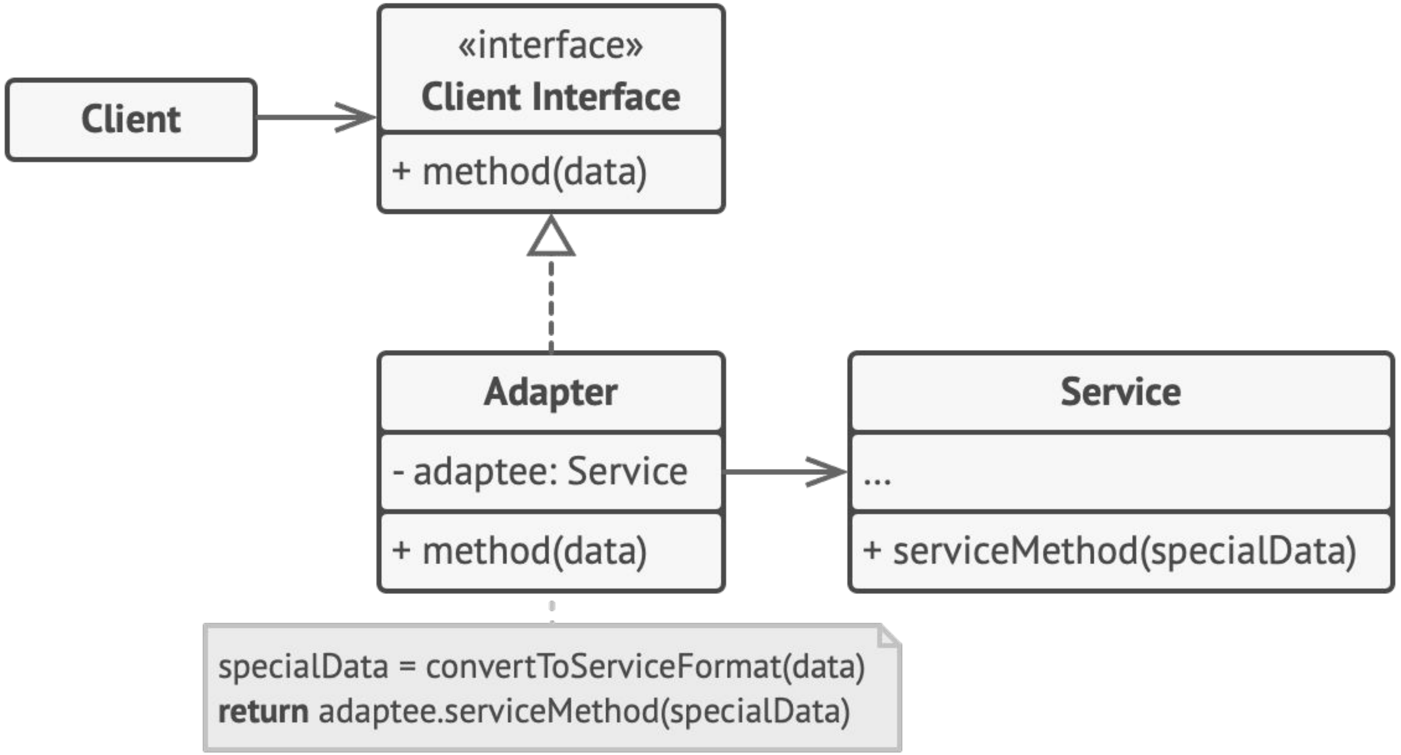
```
var lines = IOHelper.readLinesFromFile(file);
var linesObs = Observable.fromIterable(lines);
linesObs.
       map(Parser::getURLColumn).
       groupBy(...).
       sorted(comparator).
       subscribe(IOHelper.writeToFile(outFile));
```

Many more Features in Reactive Programming Libraries

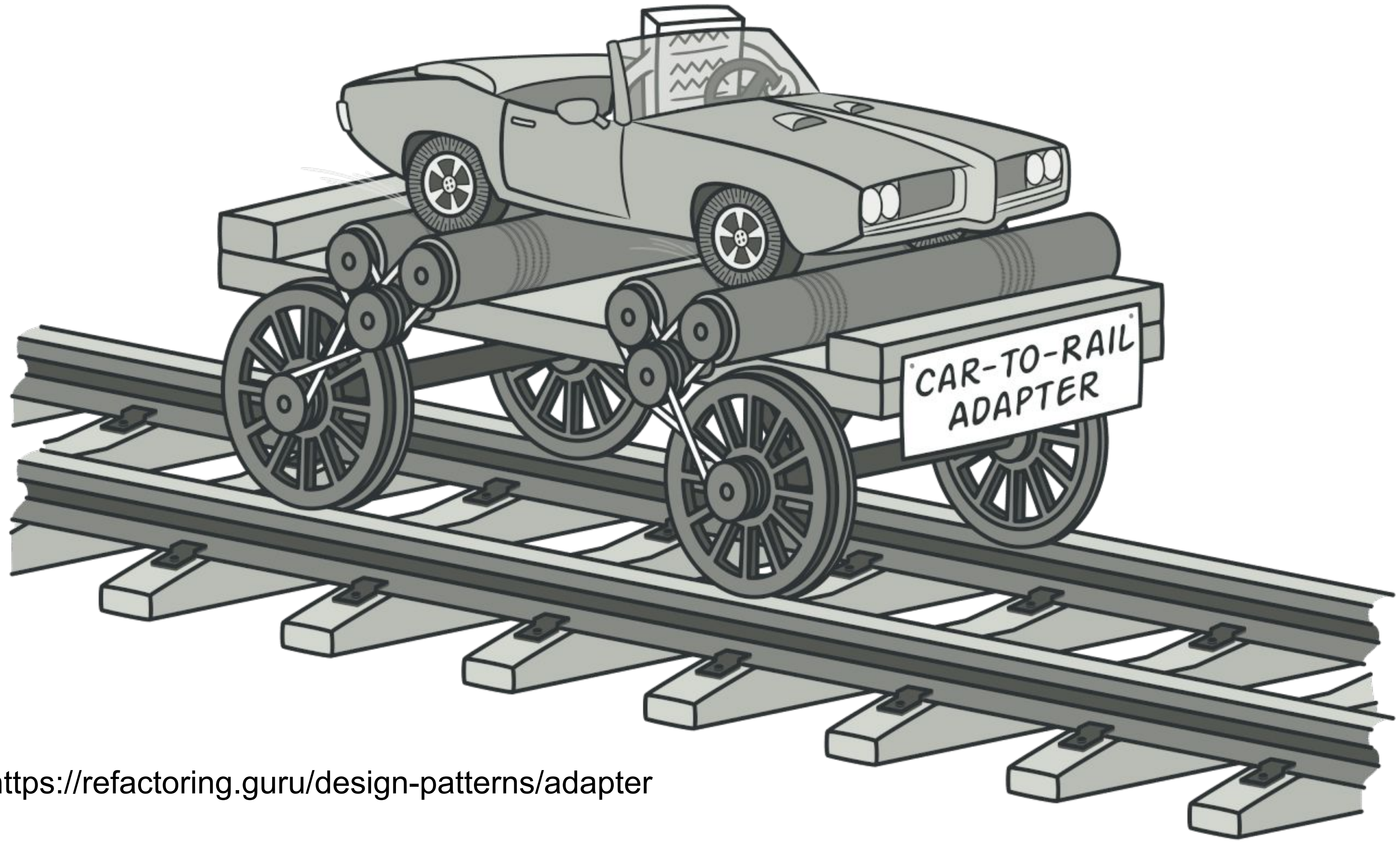
Backpressure (see last lecture)

Aside: The Adapter Pattern

The *Adapter* Design Pattern

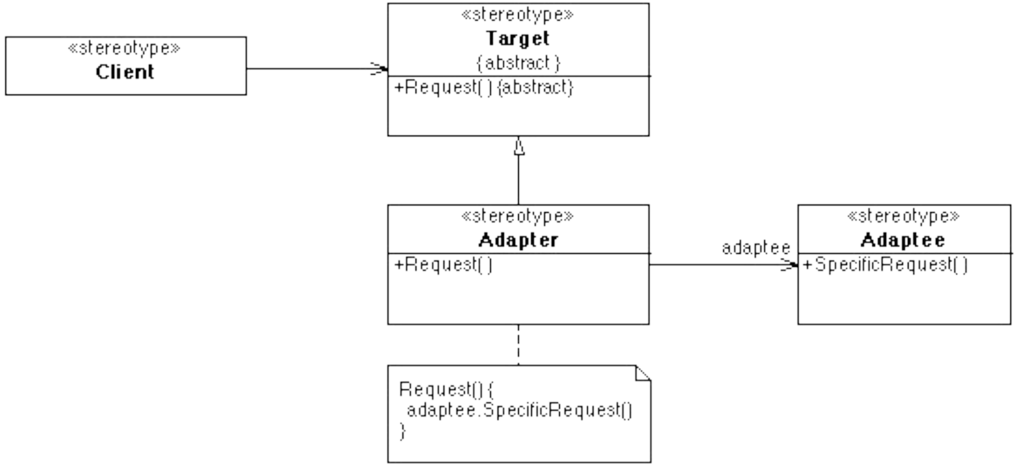


```
specialData = convertToServiceFormat(data)
return adaptee.serviceMethod(specialData)
```



https://refactoring.guru/design-patterns/adapter

The *Adapter* Design Pattern



Applicability

- You want to use an existing class, and its interface does not match the one you need
- You want to create a reusable class that cooperates with unrelated classes that don't necessarily have compatible interfaces
- You need to use several subclasses, but it's impractical to adapt their interface by subclassing each one

Consequences

- Exposes the functionality of an object in another form
- Unifies the interfaces of multiple incompatible adaptee objects
- Lets a single adapter work with multiple adaptees in a hierarchy
- -> **Low coupling, high cohesion**

Adapters for Collections/Streams/Observables

```
var lines = IOHelper.readLinesFromFile(file);
var linesObs = Observable.fromIterable(lines);
linesObs.
       map(Parser::getURLColumn).
       groupBy(...).
       sorted(comparator).
       subscribe(IOHelper.writeToFile(outFile));
```

Any others?

Façade/Controller vs. Adapter

- Motivation
  - Façade: simplify the interface
  - Adapter: match an existing interface
- Adapter: interface is given
  - Not typically true in Façade
- Adapter: polymorphic
  - Dispatch dynamically to multiple implementations
  - Façade: typically choose the implementation statically

Summary

Reactive programming decouples programs along data

- Observer pattern on steroids

New Design Pattern: Adapter

Decompose GUI from Core with Model View Controller Pattern

Brief intro to ReactJS