24-25 October, 2019

# Type-safer React & Redux applications

Jiayi Hu

## Schedule of this talk

1. Introduction to TypeScript type-system
2. Modeling domain business logic
3. Modeling UI constraints
4. Better Redux typings

## TypeScript

— A superset of JavaScript, it adds a type system

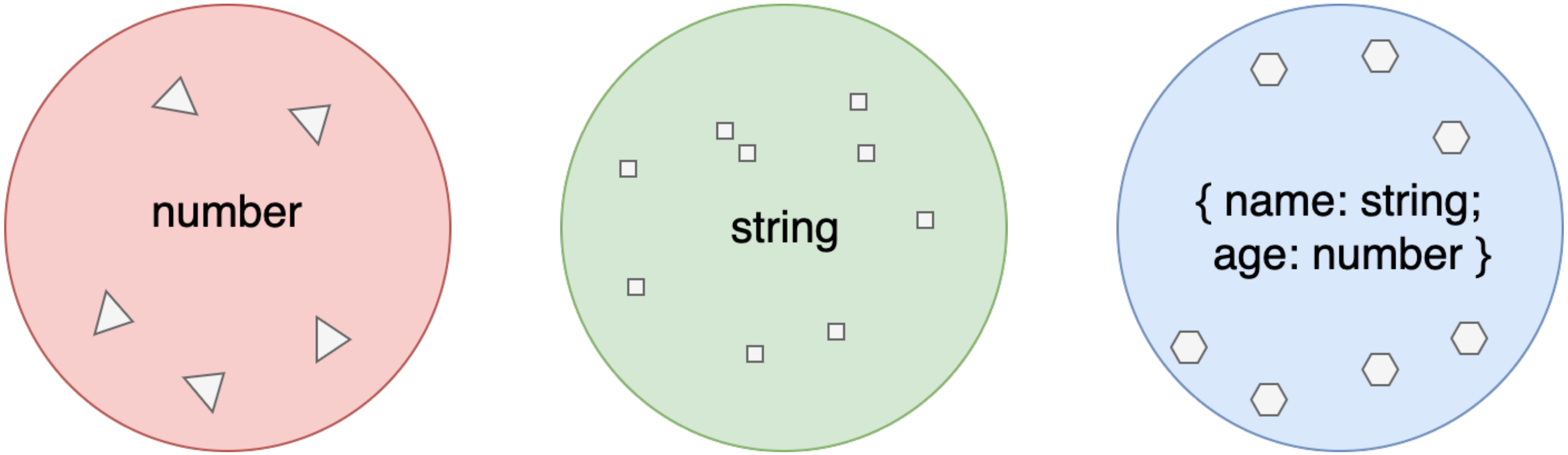
— Compiles into a target ES version

— It has a powerful language server, built-in into VSCode

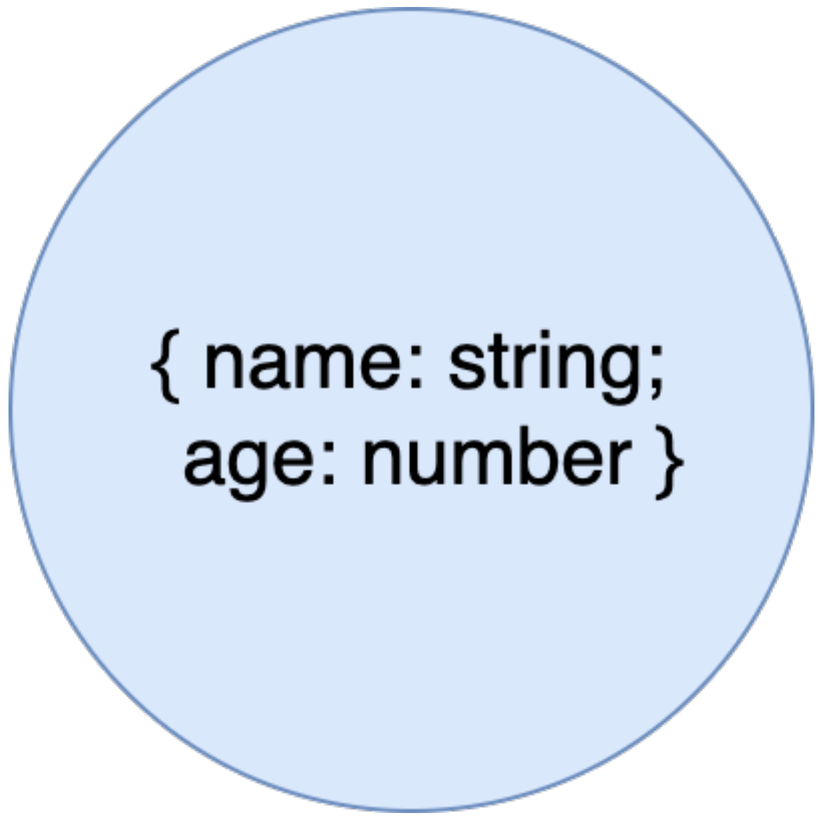
— Open source from Microsoft

TS

A type defines the set of values a variable can take.



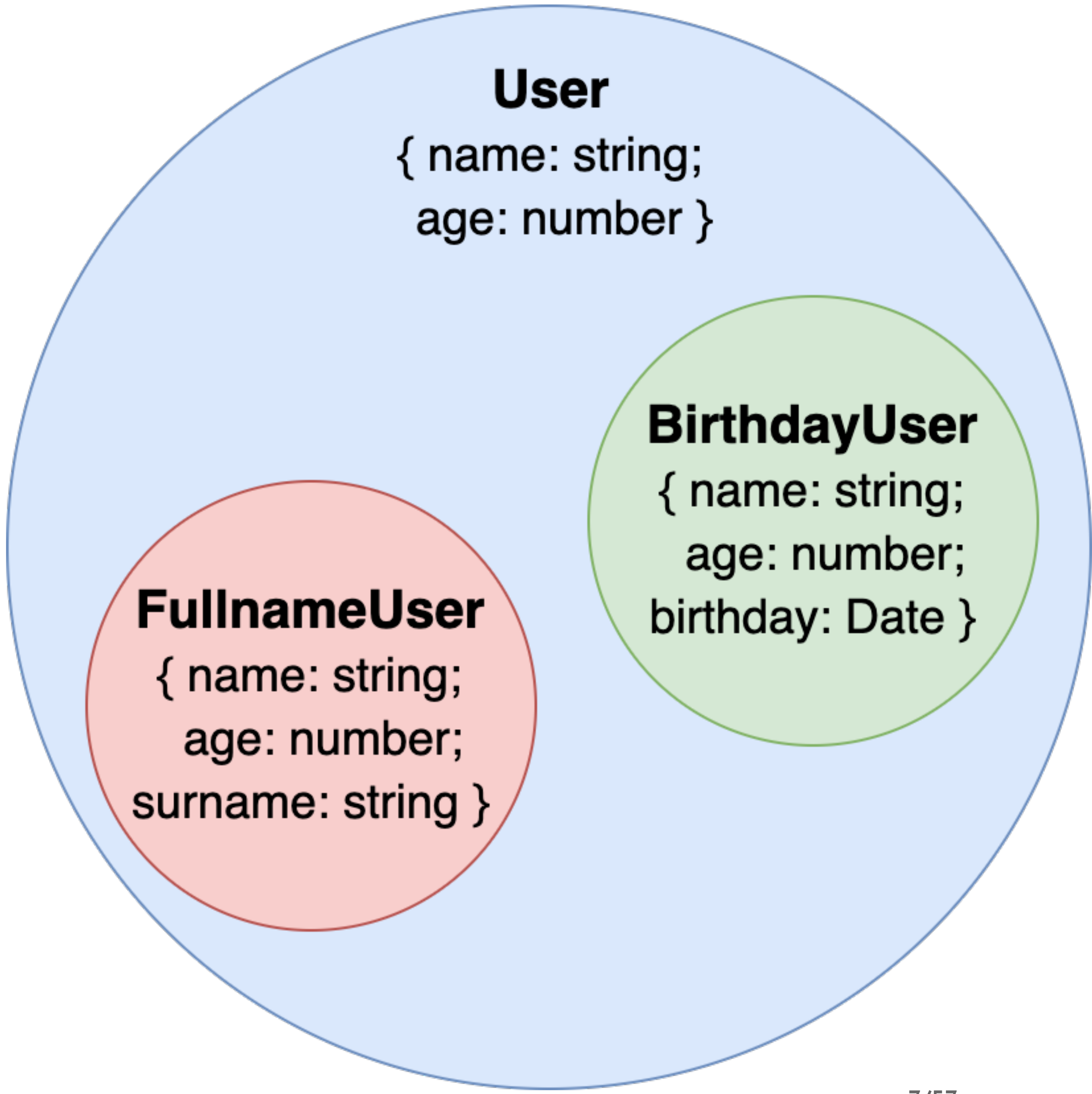
```
const age: number = 23
const name: string = 'Jiayi'
const user: { name: string, age: number } =
  { name: 'Jiayi', age: 23 }
```



```
type User = { name: string, age: number }

const user: User = { name: 'Jiayi', age: 23 }
```

## Structural subtyping

```
type User = {
  name: string,
  age: number
}

const surnameUser: User = {
  name: 'Jiayi',
  age: 23,
  surname: 'Hu'
}

const birthdayUser: User = {
  name: 'Jiayi',
  age: 23,
  birthday: new Date()
}
```



## Structural subtyping

```
function isAdultUser(user: User): boolean {
  return user.age >= 18
}

isAdultUser({ name: 'Jiayi', age: 23 }) // Okay
isAdultUser({ name: 'Jiayi', age: 23, surname: 'Hu' }) // Okay
isAdultUser({ name: 'Jiayi', age: 23, birthday: new Date() }) // Okay
```

## Origins

Types are important for compiled language: different types does not use the same amount of memory.

## Two perspectives on type errors - 1

A discrepancy between differing data types e.g. treating an `string` as `number`.

— Usually the system terminates.

## Two perspectives on type errors - 2

A logic error: an erroneous or undesirable program behaviour, a contravention of the programmer's **explicit** intent

— The system does not terminate abnormally.

## When NASA Lost a Spacecraft Due to a Metric Math Mistake

**Remember the Mars Climate Orbiter incident from 1999?**

`pounds !== kilograms`

```
// Force is also needed in real life
function getAcceleration(mass: Kg): number {}

getAcceleration(1000) // lbs
getAcceleration(453.592) // Kg
```

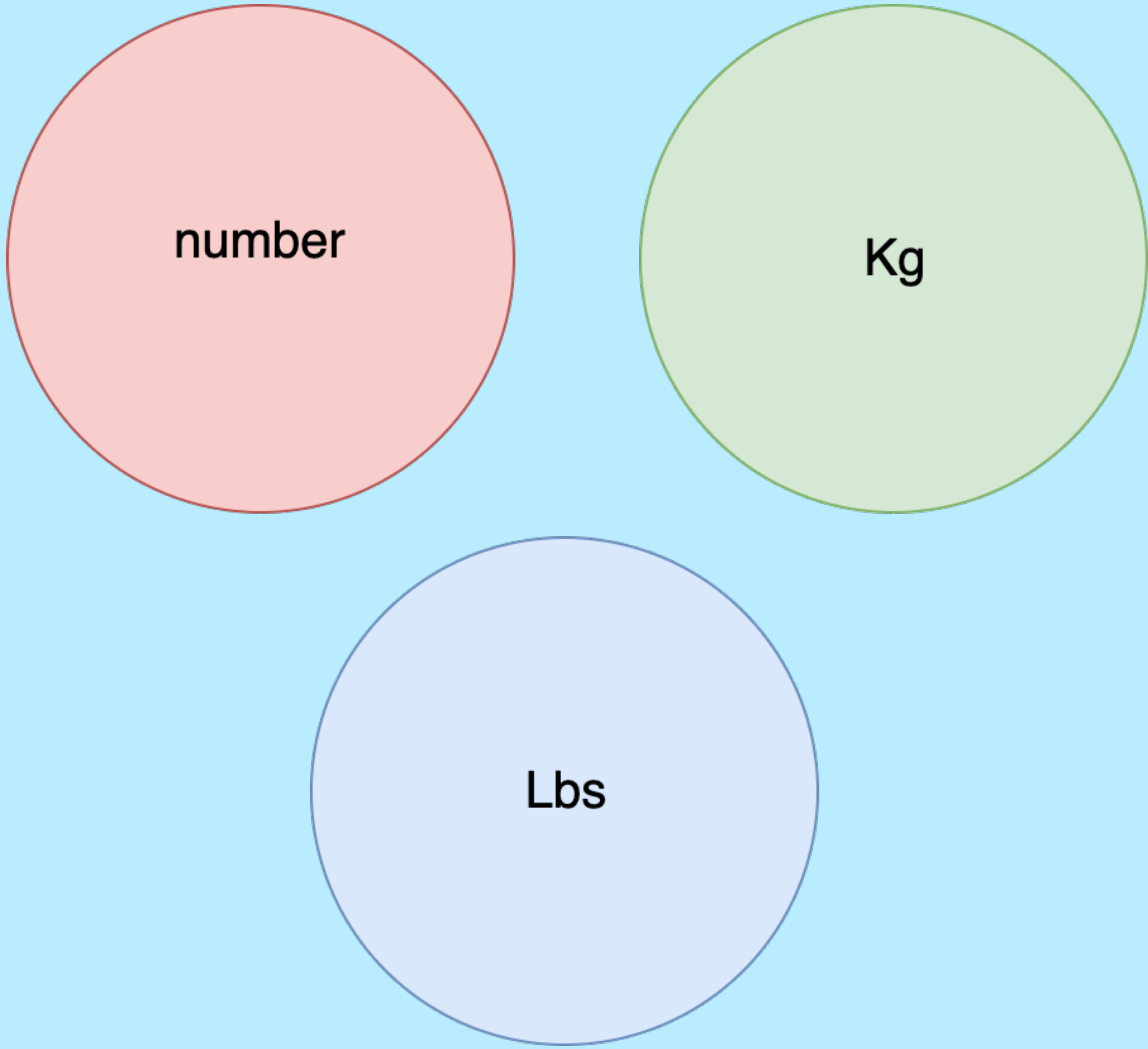
## Functional Domain modeling

## No talking about Monads.

## Types are transparent by default

```
type Lbs = number
type Kg = number

function getAcceleration(mass: Kg): number {}

const massInKg: Kg = 453.592
const massInLbs: Lbs = 1000

getAcceleration(453.592) // Okay
getAcceleration(massInKg) // Okay
getAcceleration(massInLbs) // Still okay for TS
```

## Opaque types



## Opaque types

```
// In real-life use readonly unique symbols
type Kg = { _tag: 'Kg' };
type Lbs = { _tag: 'Lbs' };

function getAcceleration(mass: Kg): number {}

const wrapAsKg = (value: number): Kg => value as any;
const wrapAsLbs = (value: number): Lbs => value as any;

const massInKg: Kg = wrapAsKg(453.592)
const massInLbs: Lbs = wrapAsLbs(1000)

getAcceleration(massInKg) // Okay
getAcceleration(453.592) // TS error
getAcceleration(massInLbs) // TS error
```

## Opaque types

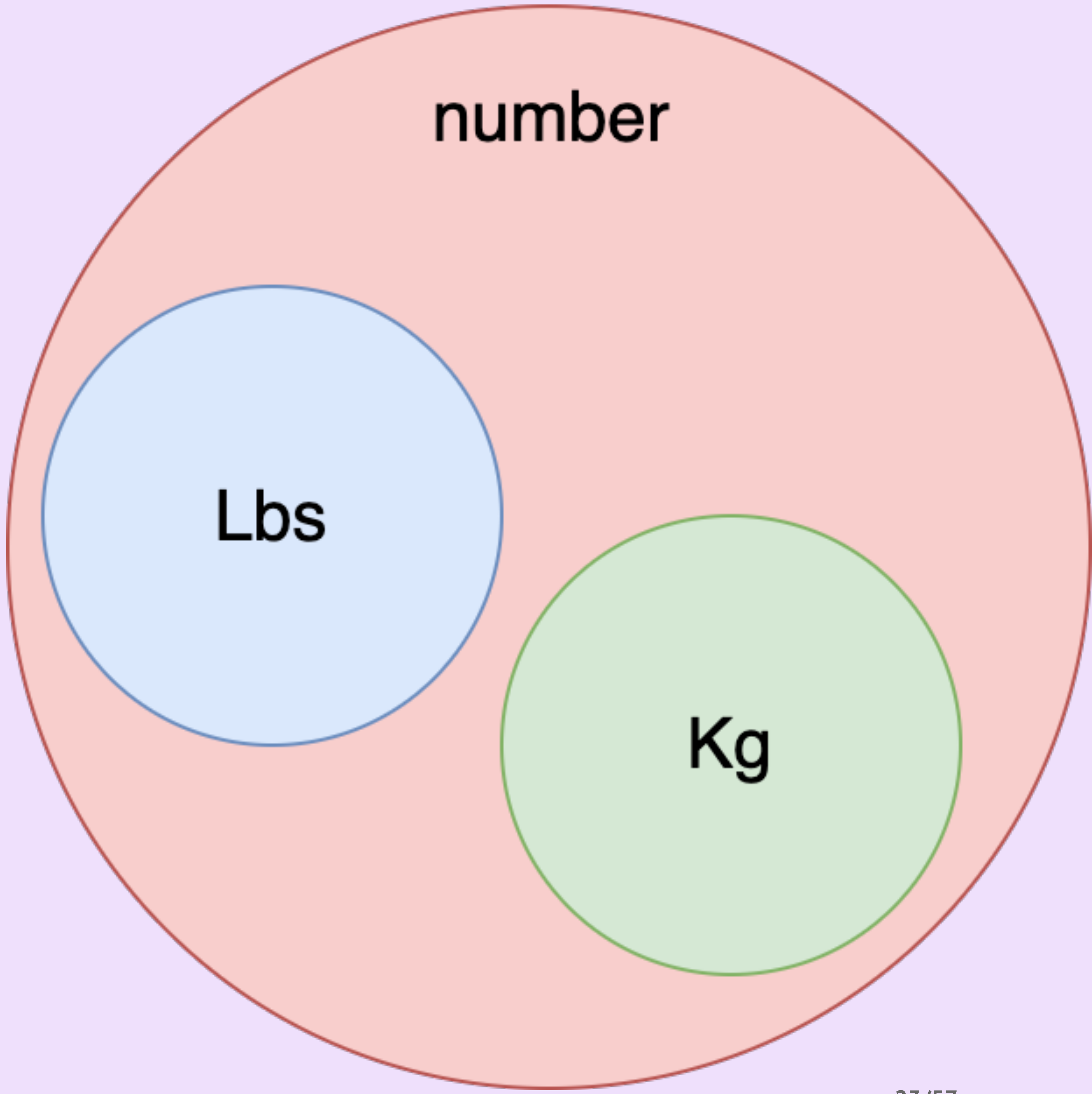
```
type Opaque<T> = { _tag: K };

type Kg = Opaque<'Kg'>;
type Lbs = Opaque<'Lbs'>;
```

## newtype-ts

```
import { Newtype, iso } from 'newtype-ts'

interface Kg extends Newtype<{ Kg: unique symbol }, number> {}

const isoKg = iso<Kg>();

const massInKg: Kg = isoKg.wrap(453.592)

function getAcceleration(mass: Kg, force: number): number {}
```

```typescript
function isPositive(value: number): boolean {
  return value > 0
}
isPositive(massInKg) // TS error

const value: number = iso.unwrap(massInKg)
isPositive(value) // Okay
```

## Branded types



## Branded types

```
type Kg = number & { _tag: 'Kg' };
type Lbs = number & { _tag: 'Lbs' };

getAcceleration(massInKg) // ok
getAcceleration(massInLbs) // TS error

isPositive(massInKg) // Okay
```

## Branded types

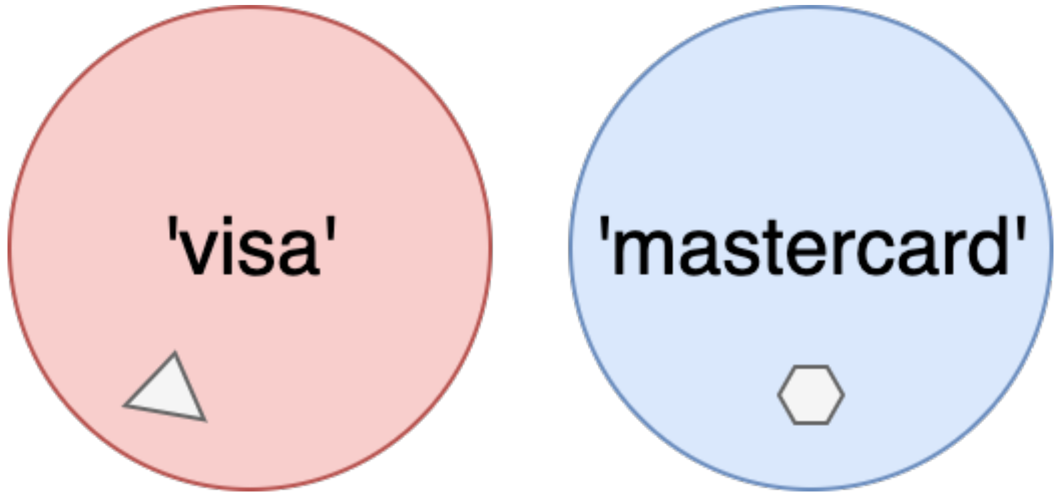
```
type Branded<K, T> = K & { _tag: T };

type Kg = Branded<number, 'Kg'>;
type Lbs = Branded<'number, Lbs'>;
```

amazon

```
export type ShippingAddress = {
  name: string;
  street: string;
  city: string;
  postalCode: string;

  isPickup: boolean;
  pickupCompany?: string;
};

export type PaymentMethod = {
  paymentMethod: string;
  cardNumber: string;
  cardExpiration: string;
  cardCode: string;
};
```

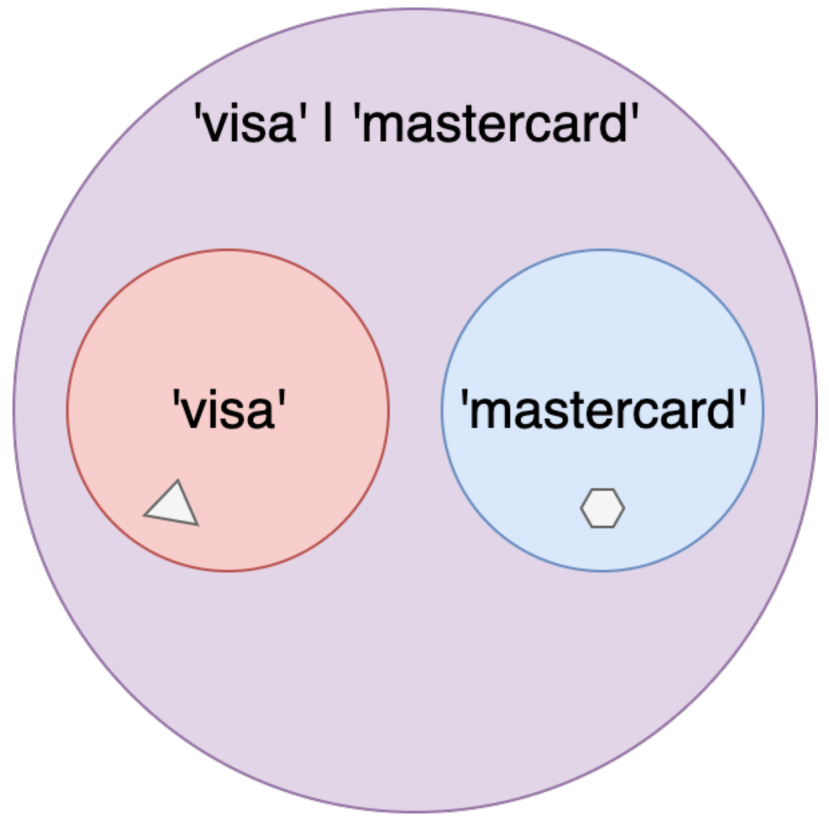
```
export type ShippingAddress = {
  name: string;
  street: string;
  city: string;
  postalCode: string;

  isPickup: boolean;
  pickupCompany?: string;
};

export type PaymentMethod = {
  paymentMethod: string;
  cardNumber: string;
  cardExpiration: string;
  cardCode: string;
};
```

## Literal types



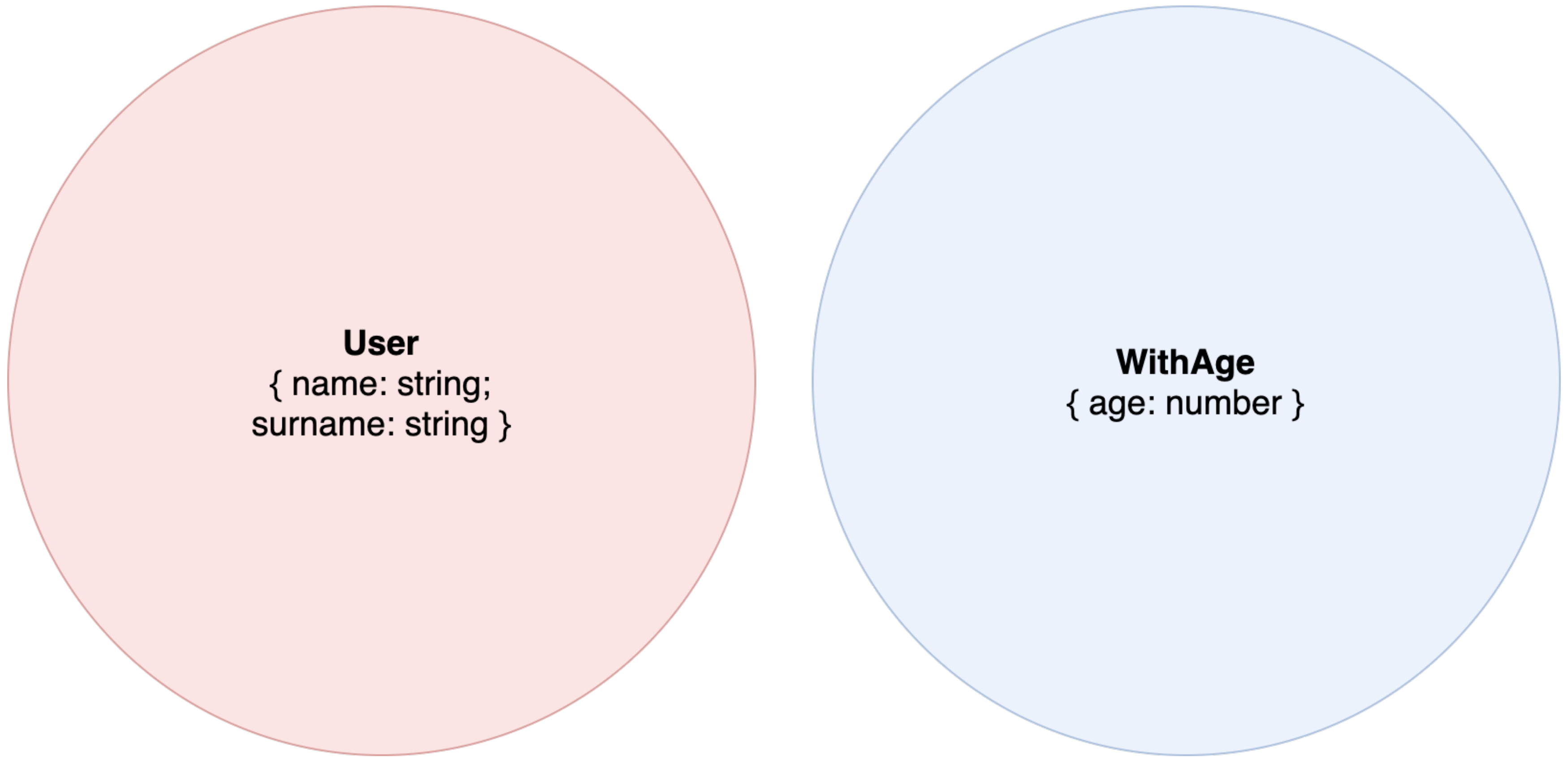
```
const visa: 'visa' = 'visa'
const mastercard: 'mastercard' = 'mastercard'
```

## Union types / Sum types



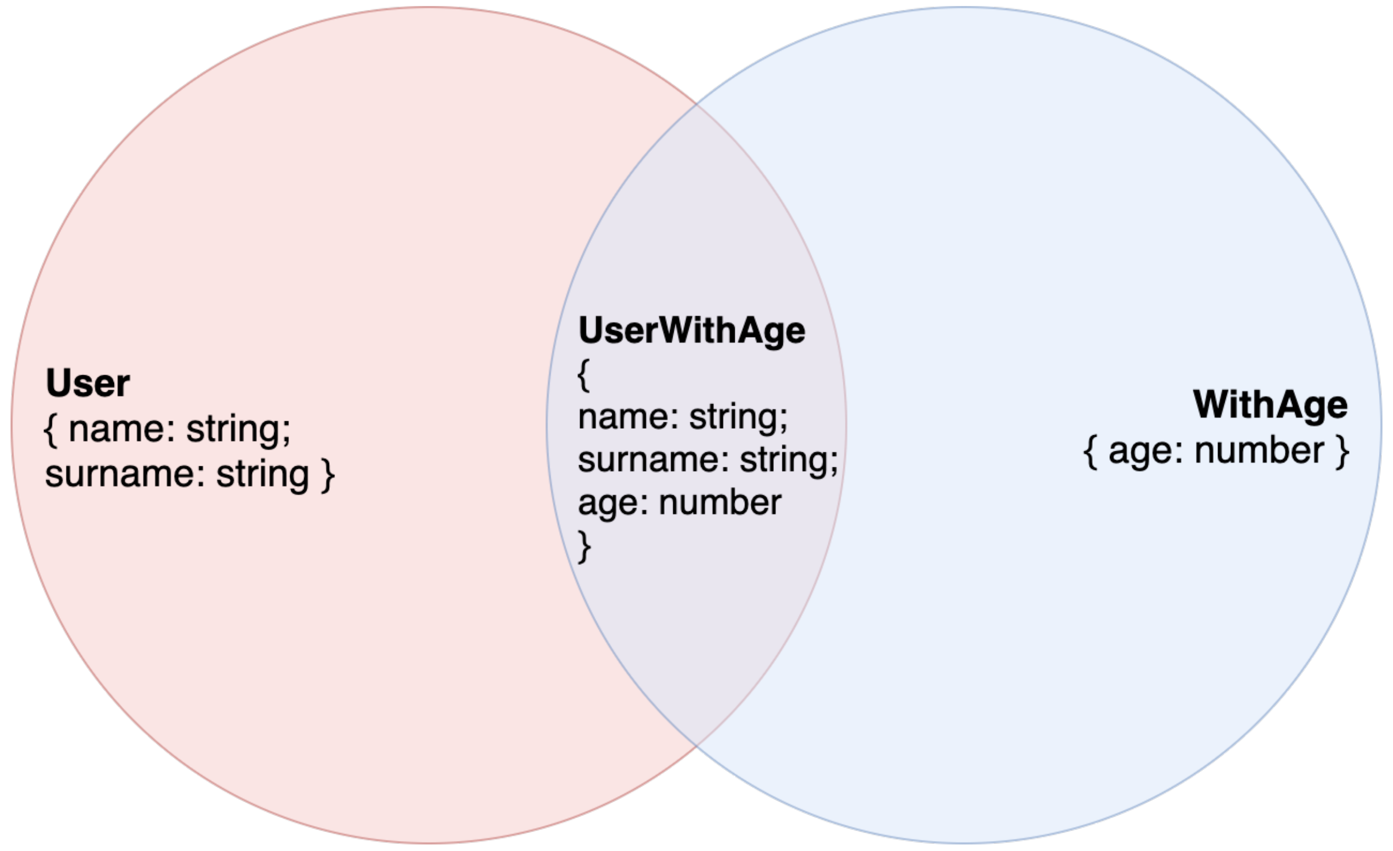
```
type PaymentMethod = 'visa' | 'mastercard'
```

## Intersection types



```
type User = { name: string, surname: string }
type WithAge = { age: number }
```

## Intersection types



```
type UserWithAge = User & WithAge
```

## Sum types - Avoid invalid states

```
type CommonInfo = {
  name: string;
  street: string;
  city: string;
  cap: string;
}

type SendToHome = CommonInfo & { type: 'SendToHome' }

type SendToPickupPoint = CommonInfo &
  { type: 'SendToPickupPoint', company: string }

type ShippingAddress = SendToHome | SendToPickupPoint
```

## Type narrowing

```
function handleShipping(address: ShippingAddress) {
  // typeof address.type === 'SendToHome' | 'SendToPickupPoint'
  console.log(address.company) // TS Error

  if (address.type === 'SendToHome') {
    // typeof address.type === 'SendToHome'
    console.log(address.company) // TS Error
    return;
  }

  // typeof address.type === 'SendToPickupPoint'

  if (address.type === 'SendToPickupPoint') {
    console.log(address.company) // Okay and with autocomplete
    return;
  }

  // typeof address.type === never
}
```

## Pattern matching [1]

```
function handleShipping(address: ShippingAddress) {
  console.log(address.company) // TS Error

  switch (address.type) {
    case 'SendToHome':
      console.log(address.company) // TS Error
      return;
    case 'SendToPickupPoint':
      console.log(address.company) // Okay and with autocomplete
      return;
    // No default case
  }
}
```

[1] tc39/proposal-pattern-matching

```
type SendToHome = CommonInfo
  & { type: 'SendToHome' }

type SendToPickupPoint = CommonInfo & {
  type: 'SendToPickupPoint',
  company: string
}

type SendToAmazonLocker = CommonInfo & {
  type: 'SendToAmazonLocker',
  code: string
}

type Pickup =
  | SendToHome
  | SendToPickupPoint
  | SendToAmazonLocker
```

```
function handleShipping(address: ShippingAddress) {
  switch (address.type) {
    case 'SendToHome':
      return;
    case 'SendToPickupPoint':
      console.log(address.company)
      return;
    // TS Error, missing 'SendToAmazonLocker' case
  }
}
```

```
type CardType = 'Visa' | 'Mastercard'
type CardPayment = {
  type: 'CardPayment';
  card: CardType;
  cardNumber: string;
  cardExpiration: string;
  cardCode: string;
};

type CashPayment = {
  type: 'CashPayment';
}

type ChequePayment = {
  type: 'ChequePayment';
}

export type PaymentMethod = CardPayment | CashPayment | ChequePayment
```

```
function render(payment: PaymentMethod) {
  switch (payment.type) {
    case 'CardPayment':
      return renderCardPayment(payment)
    case 'CashPayment':
      return renderCashPayment(payment)
    case 'ChequePayment':
      return renderChequePayment(payment)
  }
}

function renderCardPayment(cardPayment: CardPayment)
  : React.ReactElement {}
```

## UI State

Make illegal states unrepresentable

```
type Props = {
  placeholder?: string;

  isMultiple?: boolean;
  value: string | string[];
  onChange: (value: string | string[]) => void;
}

class Select extends React.Component<Props> {}
```

```
<Select
  value={['Ocean', 'Blue']} // Missing 'multiple' prop
  onChange={onChange}
/>

<Select
  isMultiple
  value="Ocean" // Should be an array
  onChange={onChange}
/>

<Select
  value="Ocean"
  onChange={(value: string | string[]) => {
    return value.toUpperCase() // TS Error
  }}
/>
```

```
type SingleValueProps = {
  placeholder?: string;
  value: string;
  onChange: (value: string) => void;
}

type MultipleValuesProps = {
  placeholder?: string;
  isMultiple: true;
  value: string[];
  onChange: (value: string[]) => void;
}

type Props = SingleValueProps | MultipleValuesProps
```

```
<Select
  value={['Ocean', 'Blue']} // TS Error
  onChange={onChange}
/>

<Select
  isMultiple
  value="Ocean" // TS Error
  onChange={onChange}
/>

<Select
  value="Ocean"
  onChange={value => value.toUpperCase()} // Okay
/>
```

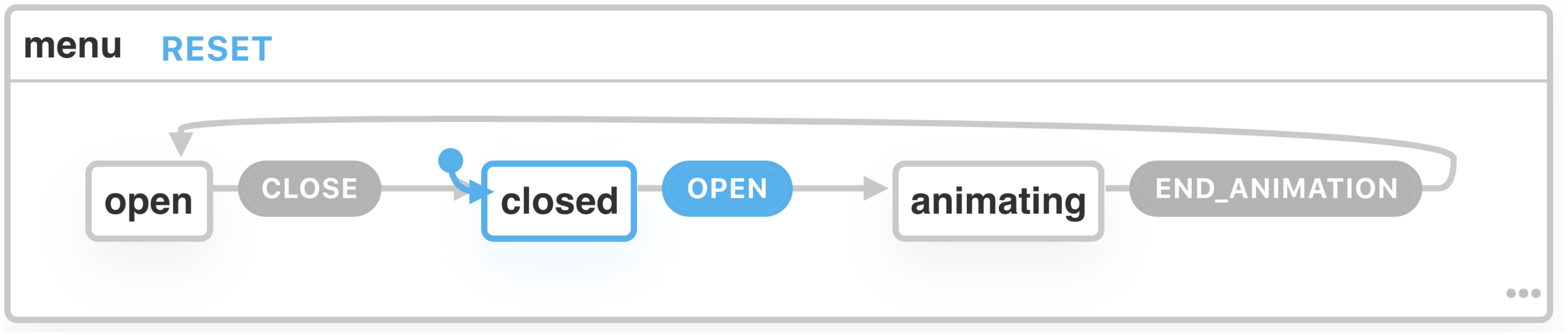
```
type CartMenuState = {
  isOpen: boolean;
  isAnimating: boolean;
  positionX: number;
}
```

```
type OpenState = { type: 'Open' }
type ClosedState = { type: 'Closed' }
type AnimatingState = { type: 'Animating', positionX: number }

type CartMenuState = OpenState | ClosedState | AnimatingState
```

```
export class CardMenu extends React.Component<{}, CardMenuState> {
  handleOpen = () => {
    this.setState({ type: 'Animating', positionX: 0 })
  }

  handleAnimating = () => {
    if (this.state.type === 'Animating') {
      this.setState({ positionX: this.state.positionX + 10 })
    }
  }

  handleAnimationEnd = () => this.setState({ type: 'Closed' })

  render() {
    ...
  }
}
```

```
export class CardMenu extends React.Component<{}, CardMenuState> {
  render() {
    switch (this.state.type) {
      case 'Open':
        return this.renderOpen();
      case 'Closed':
        return this.renderClosed();
      case 'Animating':
        return this.renderAnimating();
    }
  }
}
```

# Finite state machine



## Use cases for Sum types in React

— Drag & Drop

— Editable components

— Render props

## Type narrowing Redux actions

```
type LoginAction = {
  type: 'LOGIN_USER',
  payload: { username: string, role: string }
};
type LogoutAction = { type: 'LOGOUT_USER' };
type UpdateUserAction = { type: 'UPDATE_USER' };
type OtherActions = { type: '__OTHER_ACTIONS__' }

type Action =
  | LoginAction
  | LogoutAction
  | UpdateUserAction
  | OtherActions
```

## Type narrowing Redux actions

```
function reducer(state: State = initialState, action: Action) {
  switch (action.type) {
    case 'LOGIN_USER':
      return { ...state, username: action.payload.username }
    case 'LOGOUT_USER':
      return { ...state, username: action.payload.username } // TS error
    default:
      return state
  }
}
```

## Conclusion

- Opaque/Branded types
- Intersection types
- Sum types

## types, types, types …

## Homework

1. Try to solve the same issues with OOP
2. Type checking and unit/integration testing

## Jiayi Hu

Front-end consultant, based at Padova (IT).

{codemotion}

We code the future. Together

24-25 October, 2019