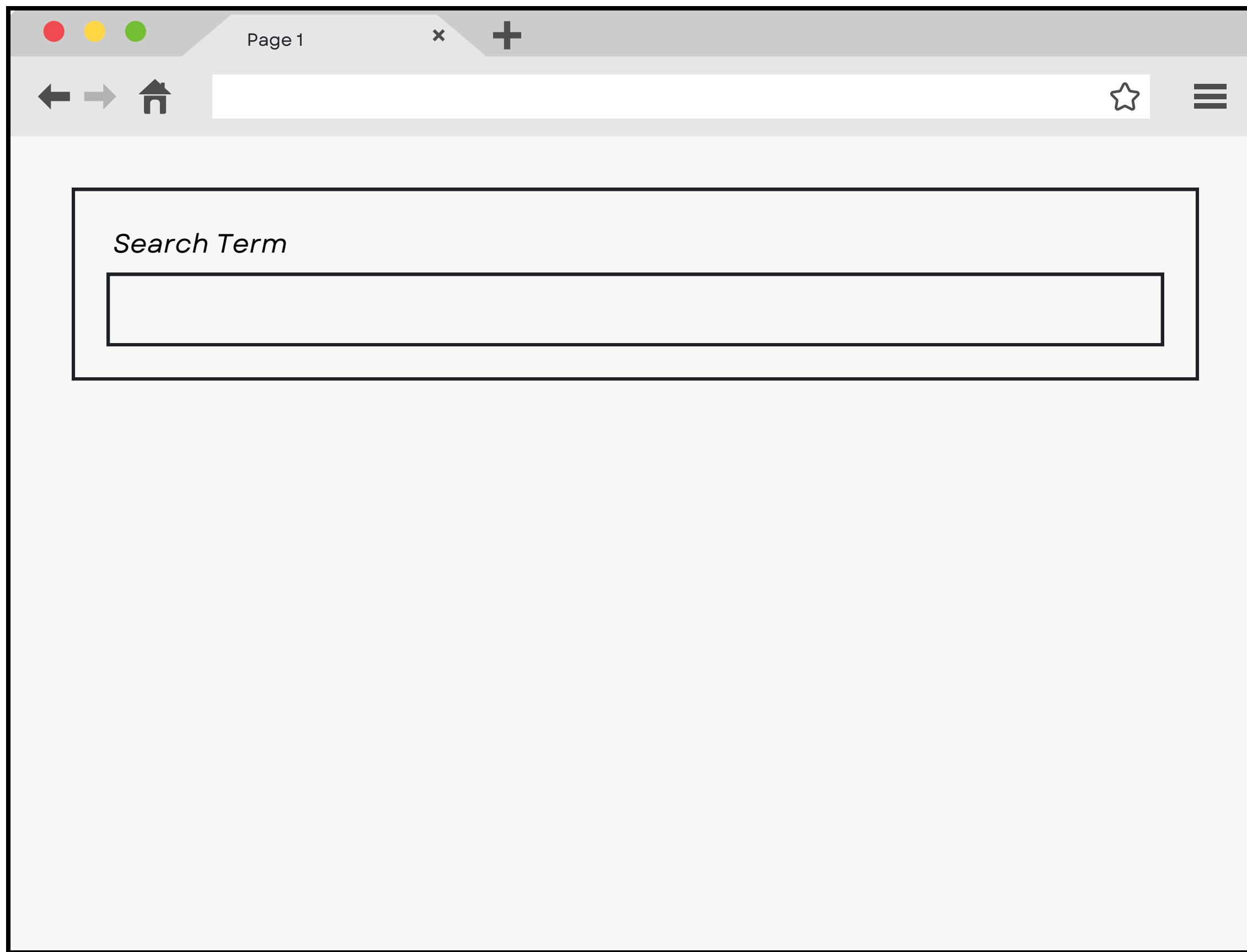
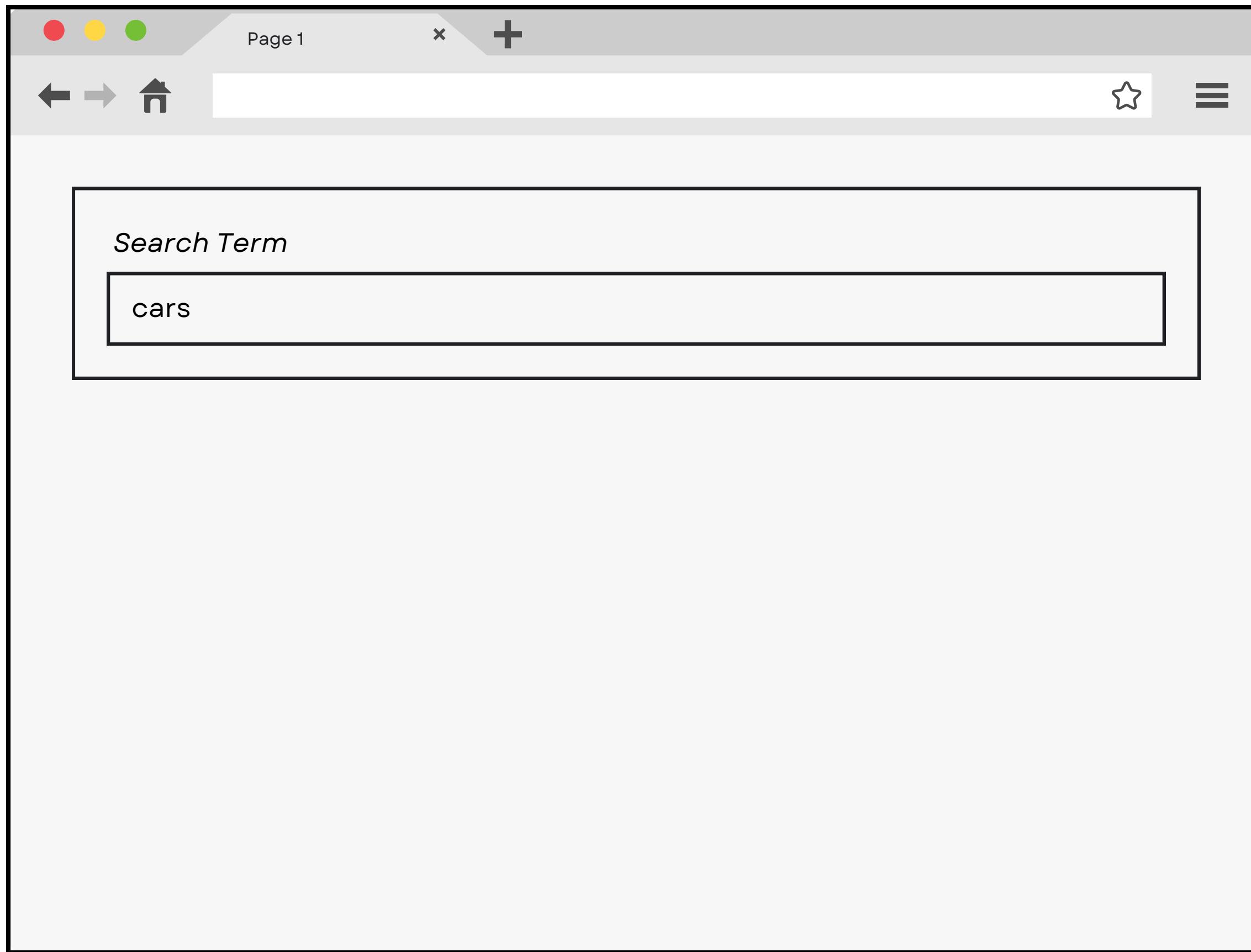
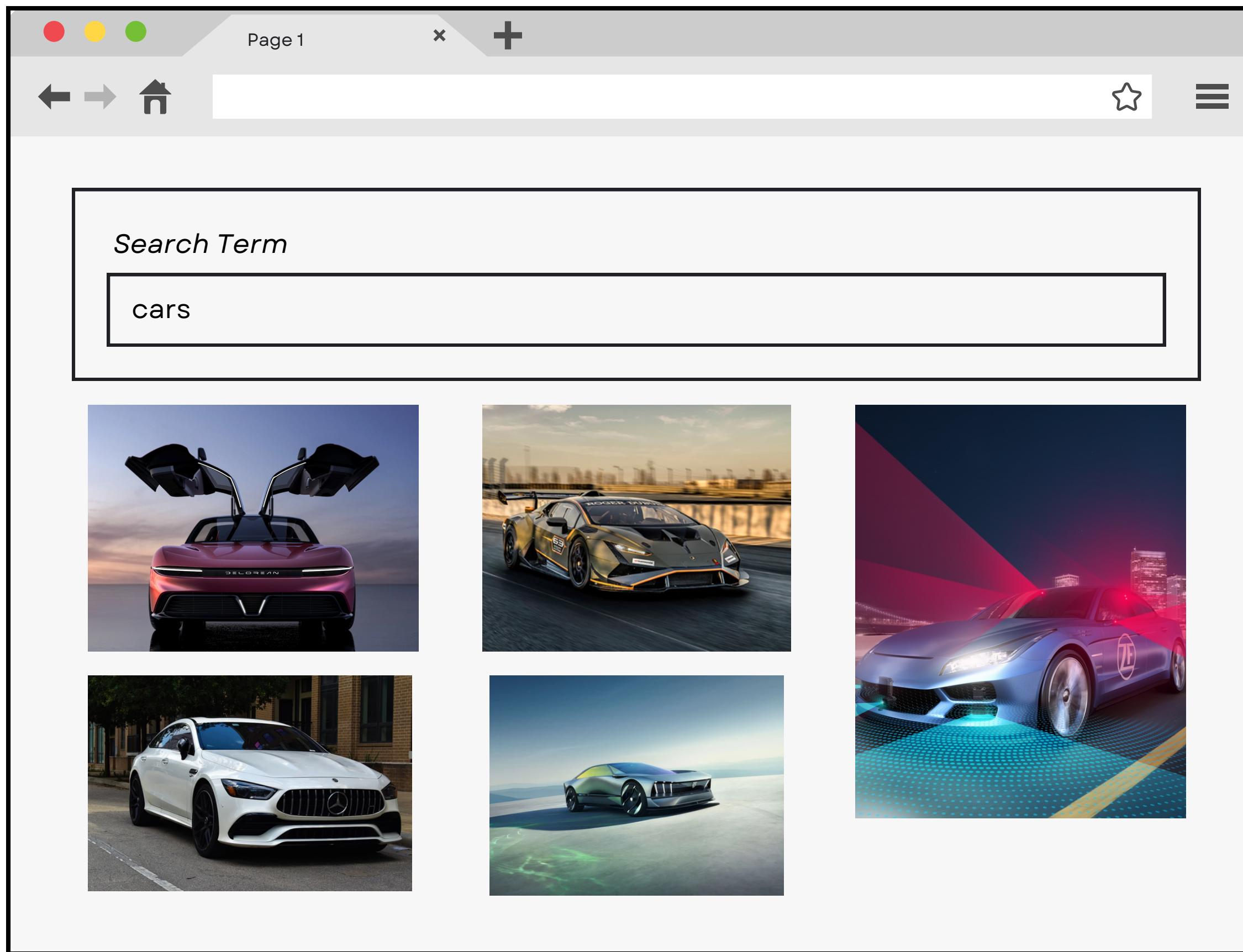


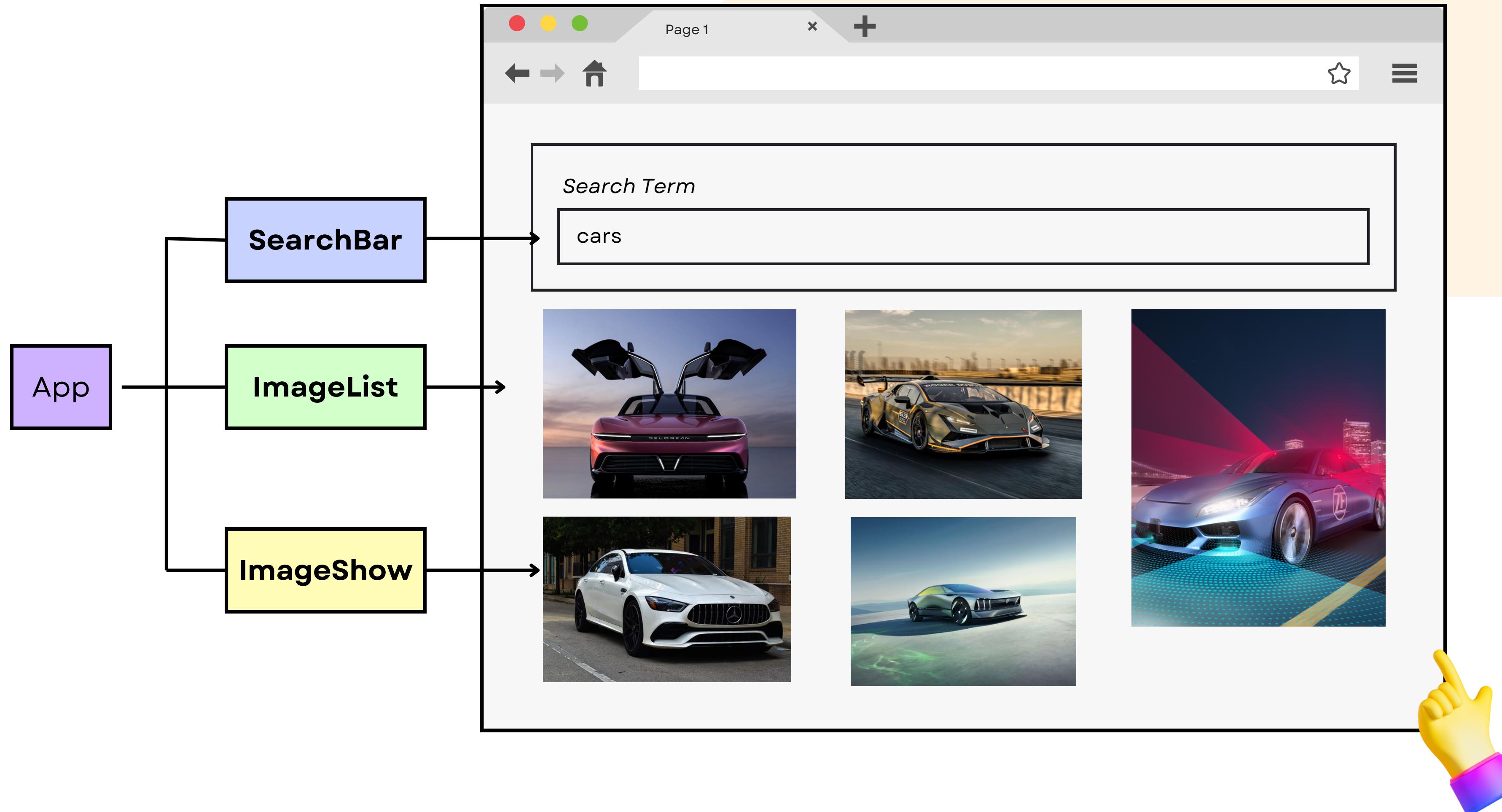
# USING AN API WITH REACT

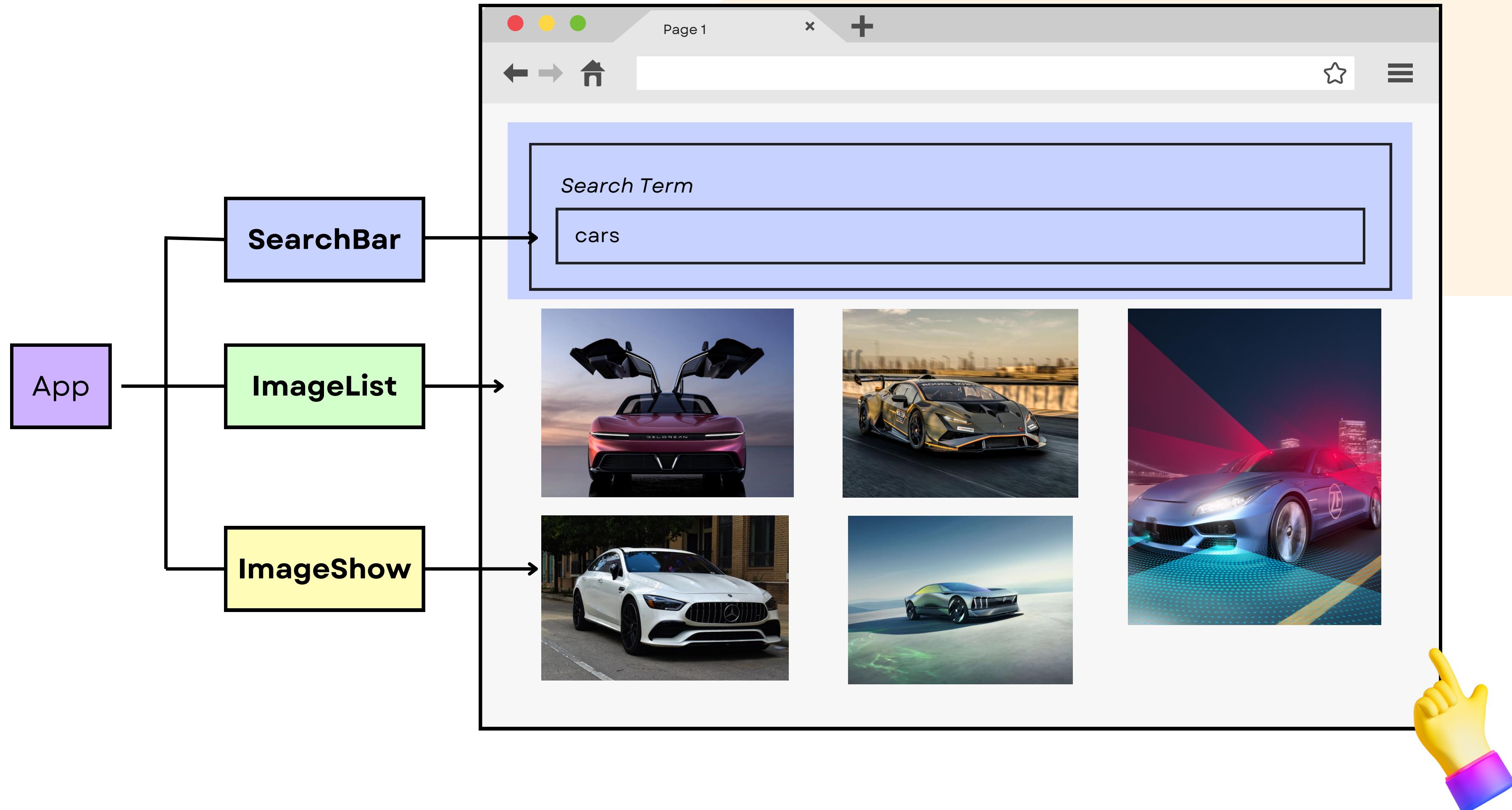


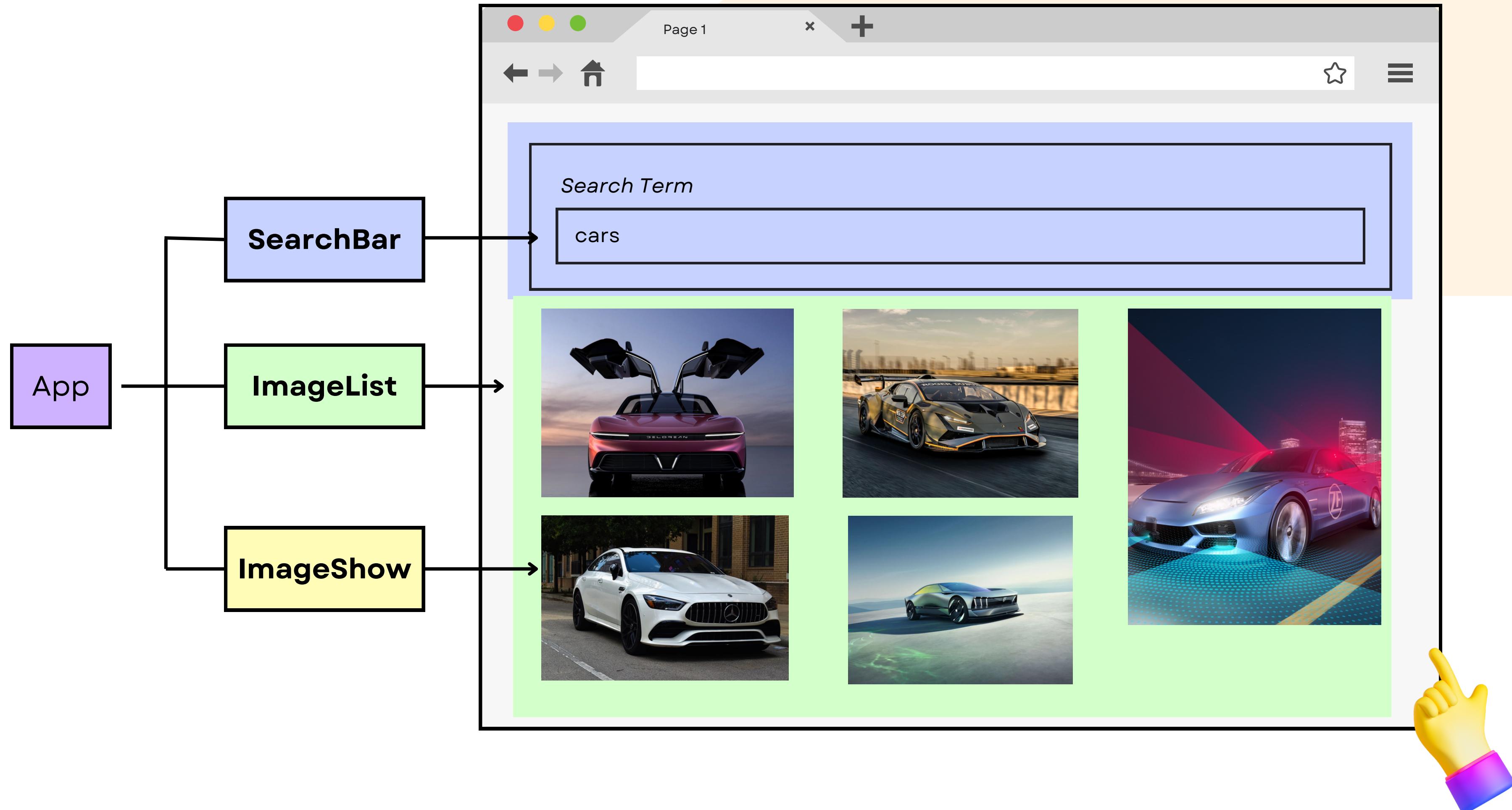


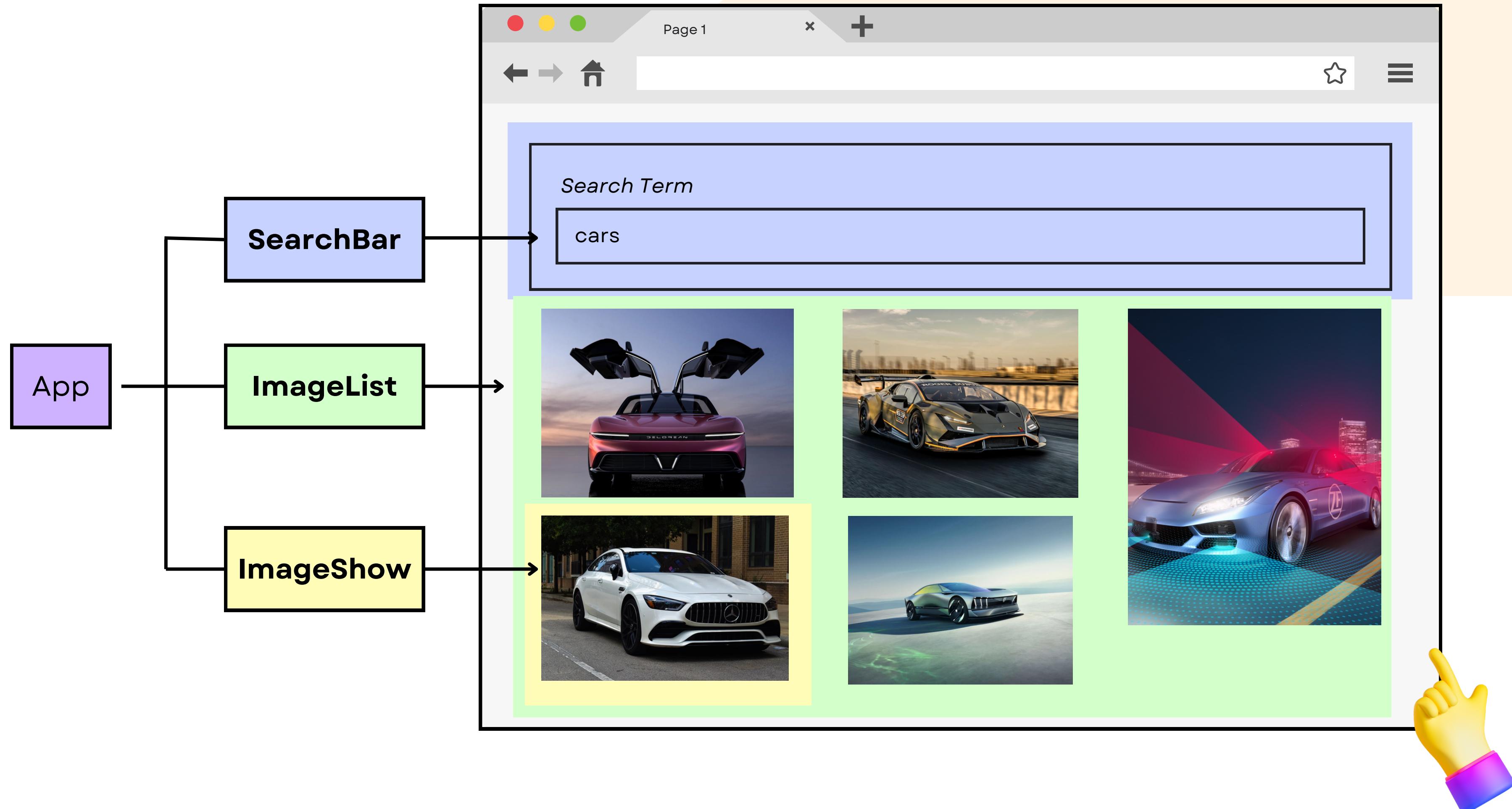


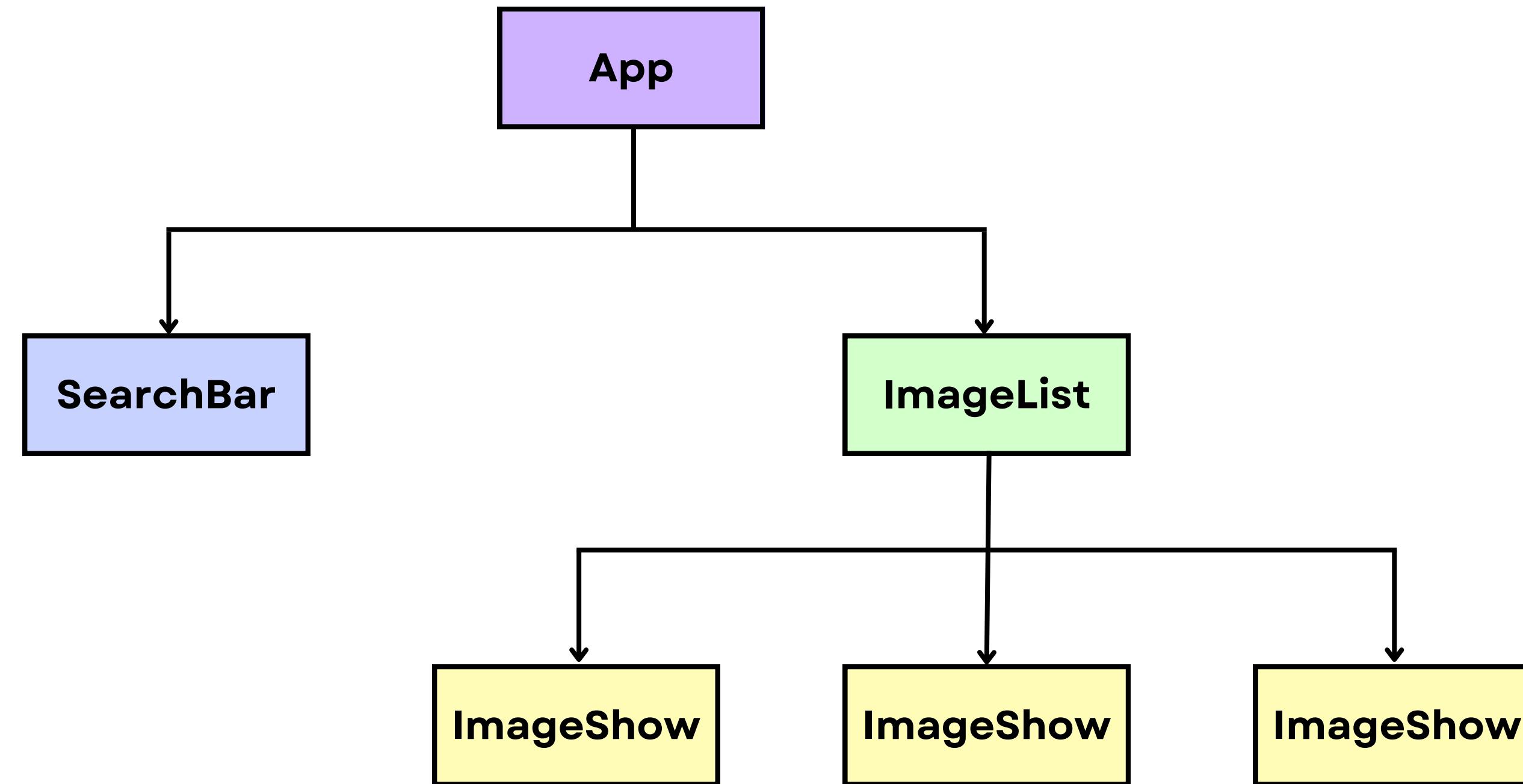


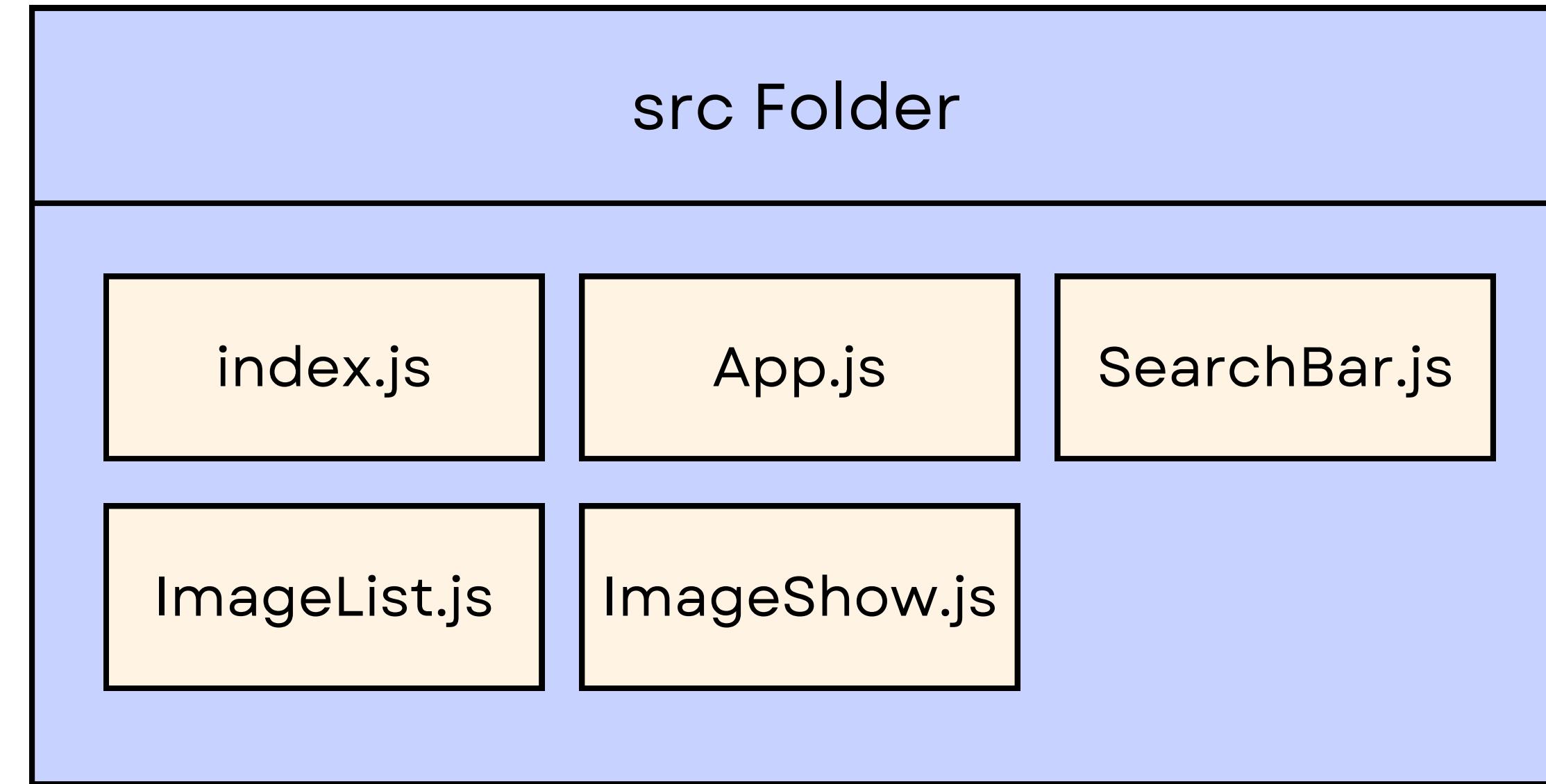












## src Folder

index.js

SearchBar.js

App.js

ImageShow.js

ImageList.js

components Folder

## src Folder

index.js

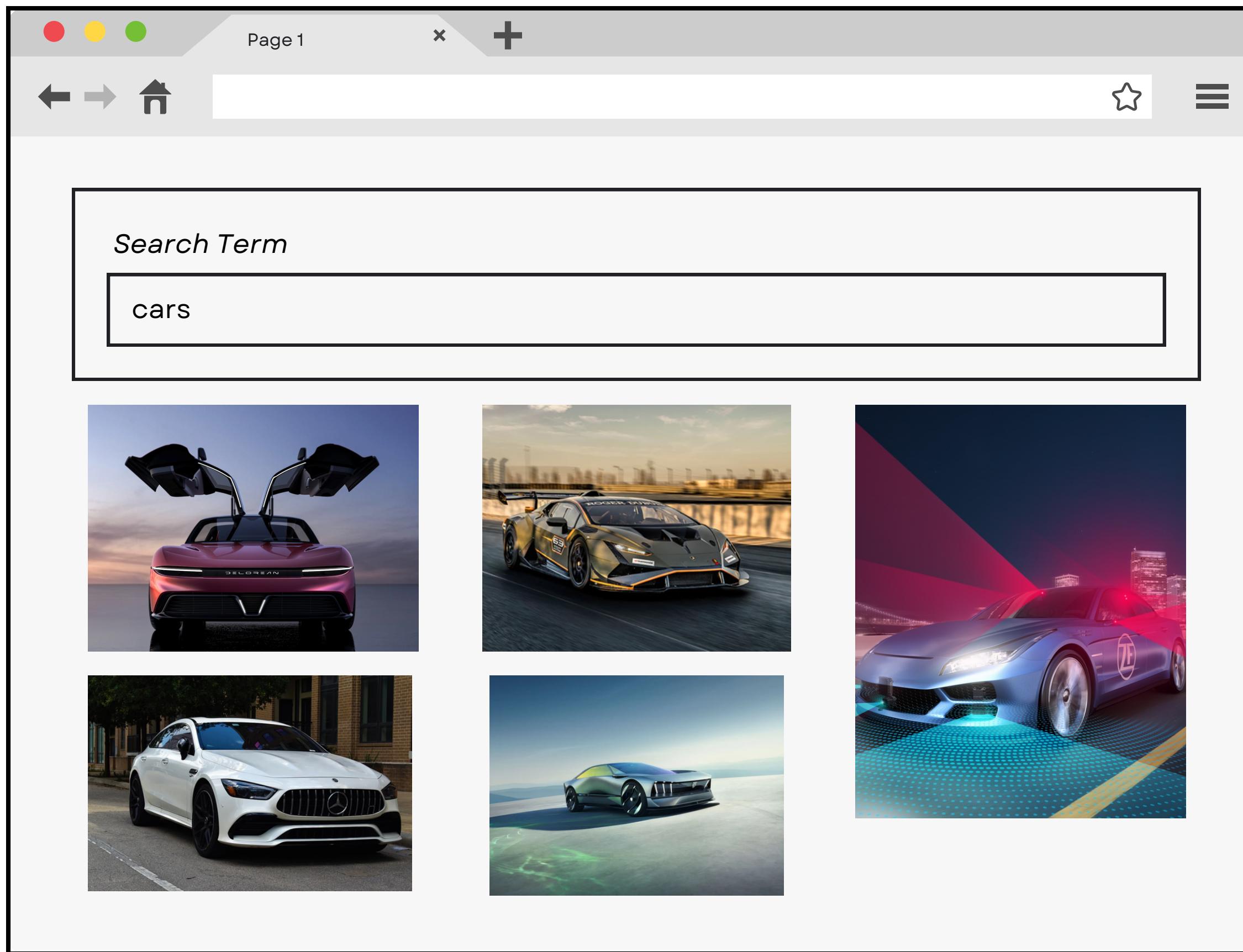
App.js

### components Folder

SearchBar.js

ImageShow.js

ImageList.js



# Our App

## Request

Can you give me some  
images related to the term  
'cars'?

# Unsplash API

## Response

Here is an array of objects.  
Each object has info about  
an image

# Our App

## Request

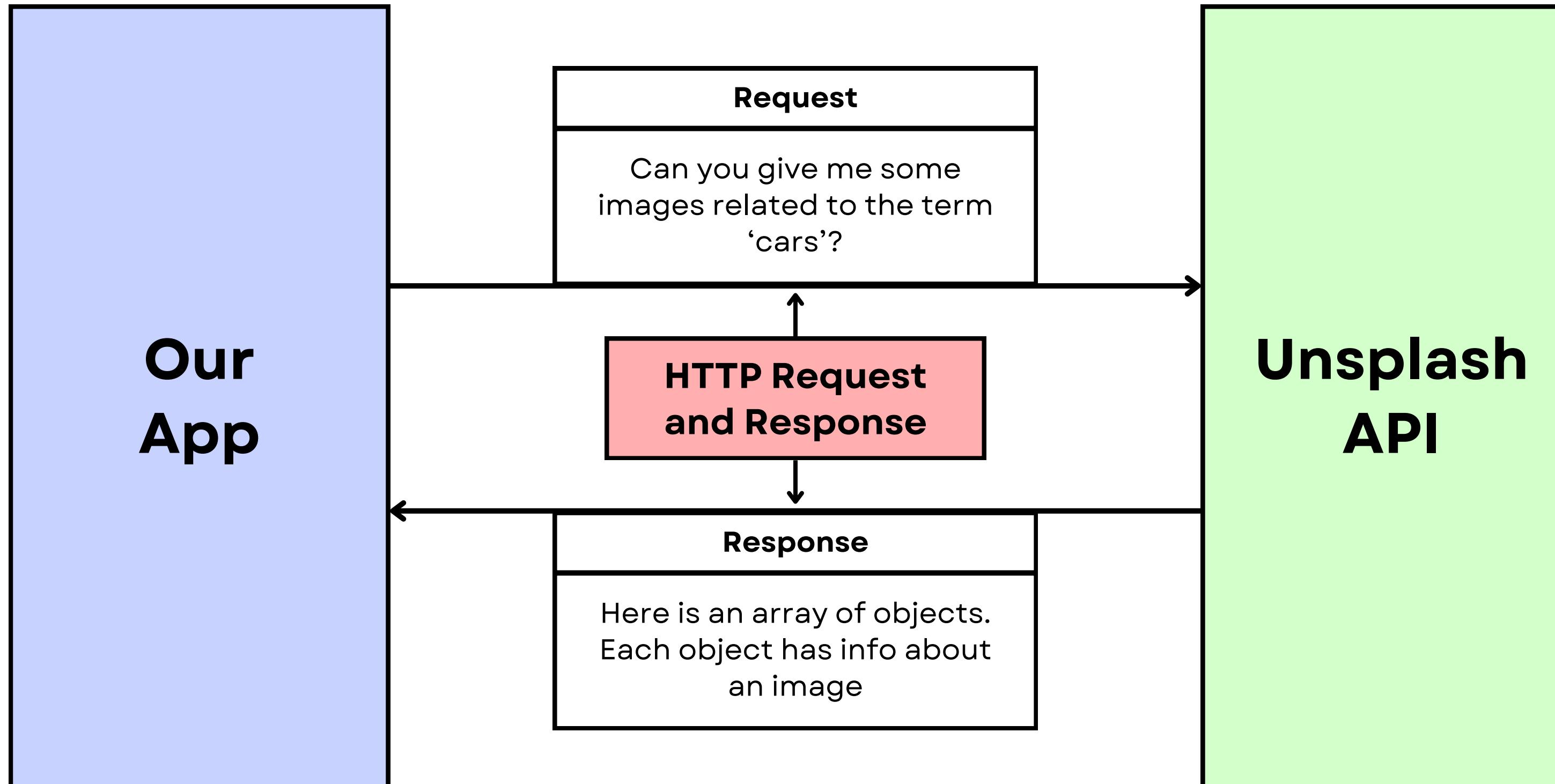
Can you give me some images related to the term 'cars'?

## HTTP Request and Response

## Response

Here is an array of objects. Each object has info about an image

# Unsplash API





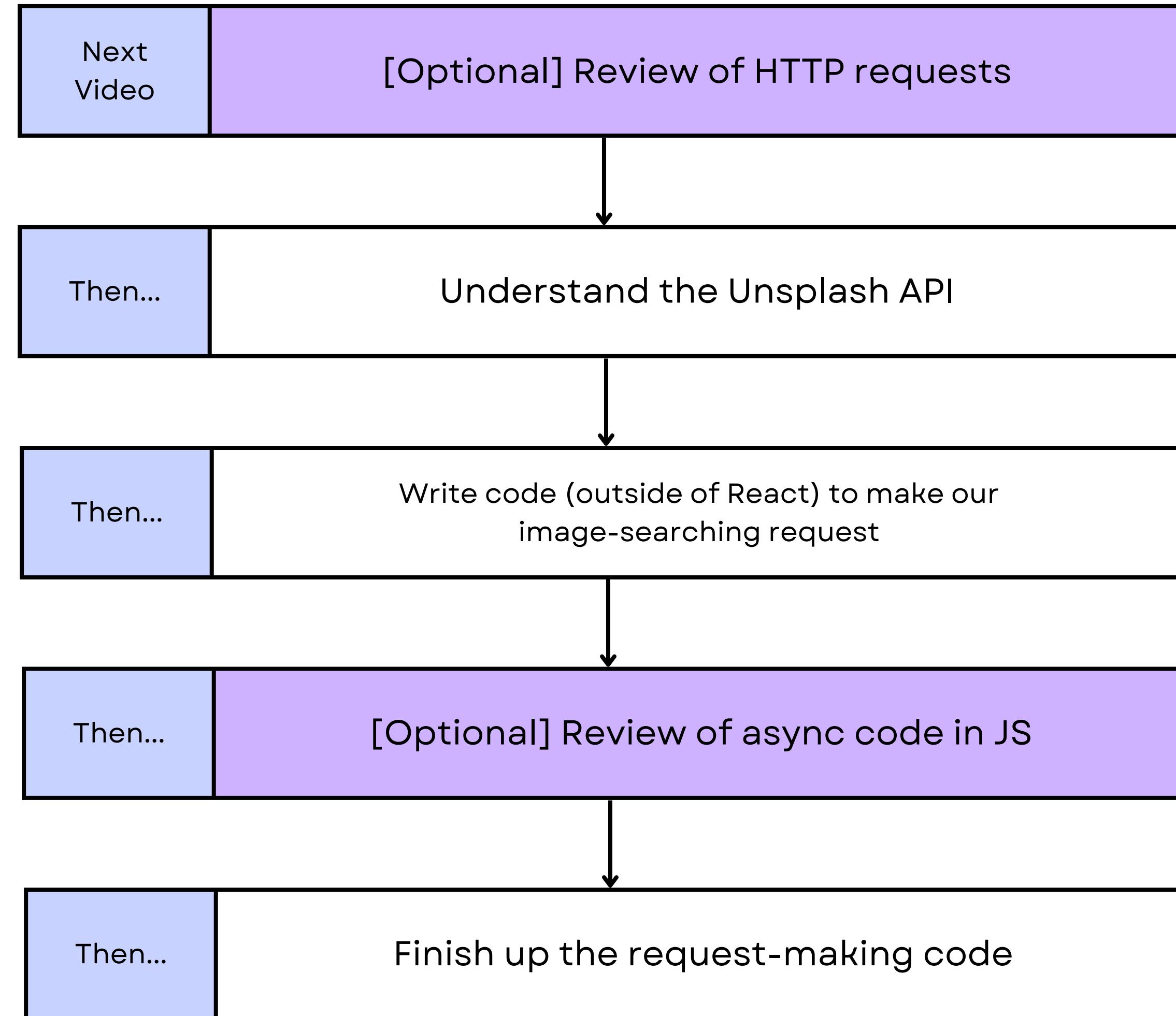
React has no tools, objects, functions for making  
HTTP requests

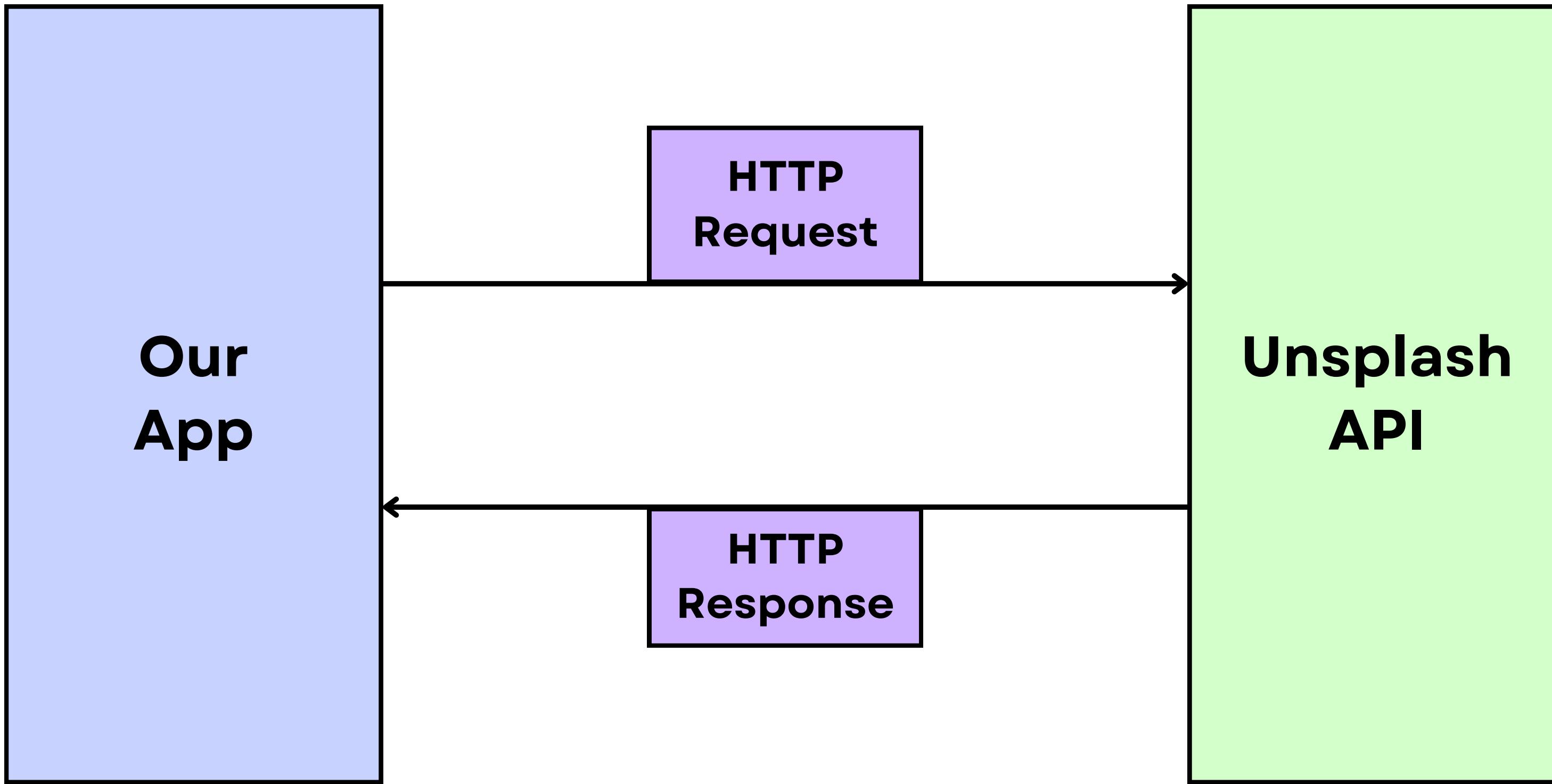


React *only* cares about showing content and  
handling user events



This is kind of good! We can write a lot of business  
logic + data fetching without worrying about React!





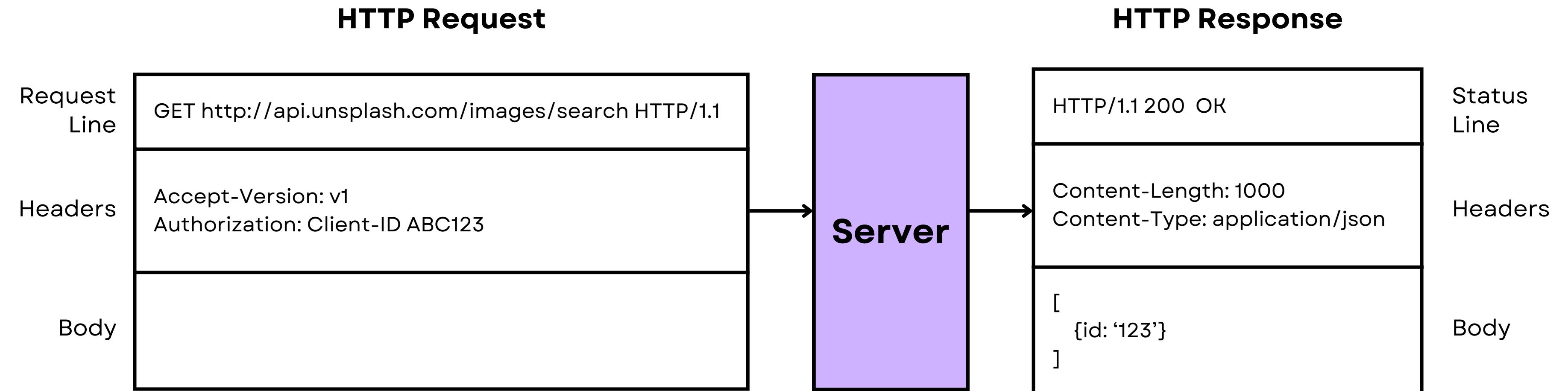
React has no tools, objects, functions for making  
HTTP requests



React *only* cares about showing content and  
handing user events



This is kind of good! We can write a lot of business  
logic + data fetching without worrying about React!



**Method**  
Indicates the general goal of the request

## HTTP Request

Request Line  
`GET http://api.unsplash.com/images/search HTTP/1.1`

Headers  
`Accept-Version: v1`  
`Authorization: Client-ID ABC123`

Body

## HTTP Response

Status Line  
`HTTP/1.1 200 OK`

Headers  
`Content-Length: 1000`  
`Content-Type: application/json`

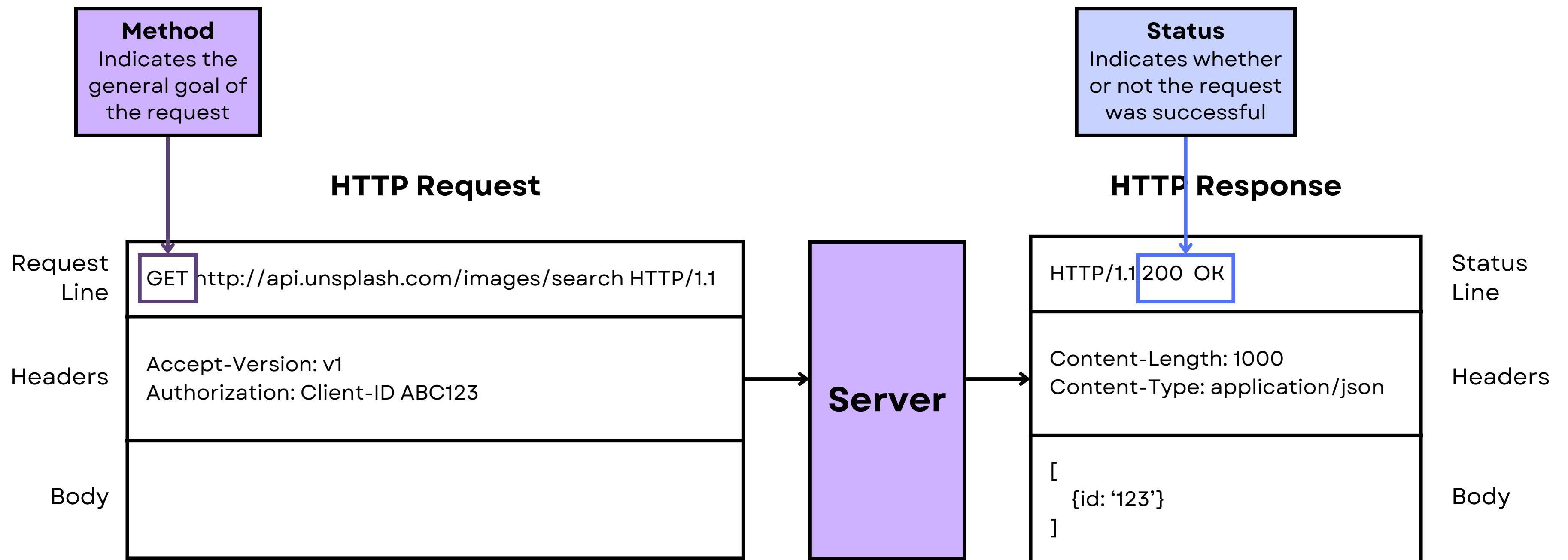
Body  
`[`  
`{id: '123'}`  
`]`

**Server**



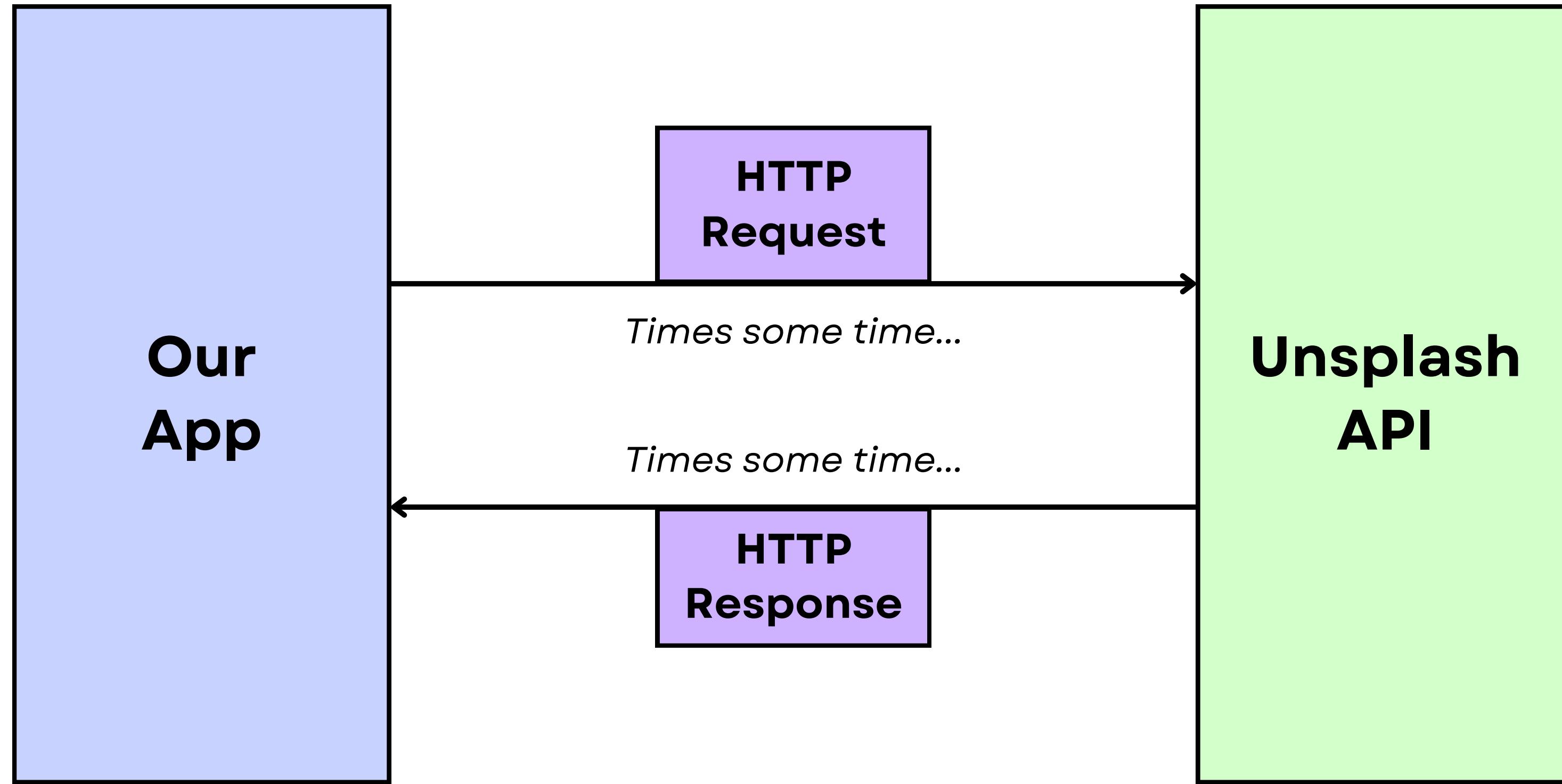
# Some HTTP Methods

GET	Get some information from the server
POST	Tell the server to create some new record
PUT	Completely update an existing record
PATCH	Partially update n existing record
DEL	Delete a record



# Some HTTP Methods

200	Request was successful, here's the data you were looking for
201	Record was created
204	Record was deleted
301	URL you made request to has changed
400	Something about your request is bad (incorrect syntax or similar)
401	Unauthorized. You must provide authentication details
403	Forbidden. You aren't allowed to access this
404	Not Found. The thing you were looking for wasn't found
500	Internal Server Error. Something on the server went wrong



*Time*



User enters a search term and presses enter

Make a request to the API

*Times passes...*

Get a response!

Use the response in some way

JS starts the request...

...then *instantly* runs this line

```
const fetchData = () => {  
  const response = makeRequest();  
  
  console.log(response);  
}
```

### Probably not good!

JS doesn't automatically pause when you make a request  
Response hasn't been received yet!



**Probably need something like this.**  
Don't try to work with the response until it has  
actually been received.

**[unsplash.com/developers](https://unsplash.com/developers)**

*Unsplash API sign up +  
documentation*

Sign up for an account



Create an ‘app’ to get an Access Key



Take a look at docs to understand  
the request we will make

# Our App

## HTTP Request

```
GET http://api.unsplash.com/search/photos?query=forests  
Authorization: Client-ID abc123
```

# Unsplash API

## Response

```
Here is an array of objects.  
Each object has info about  
an image
```

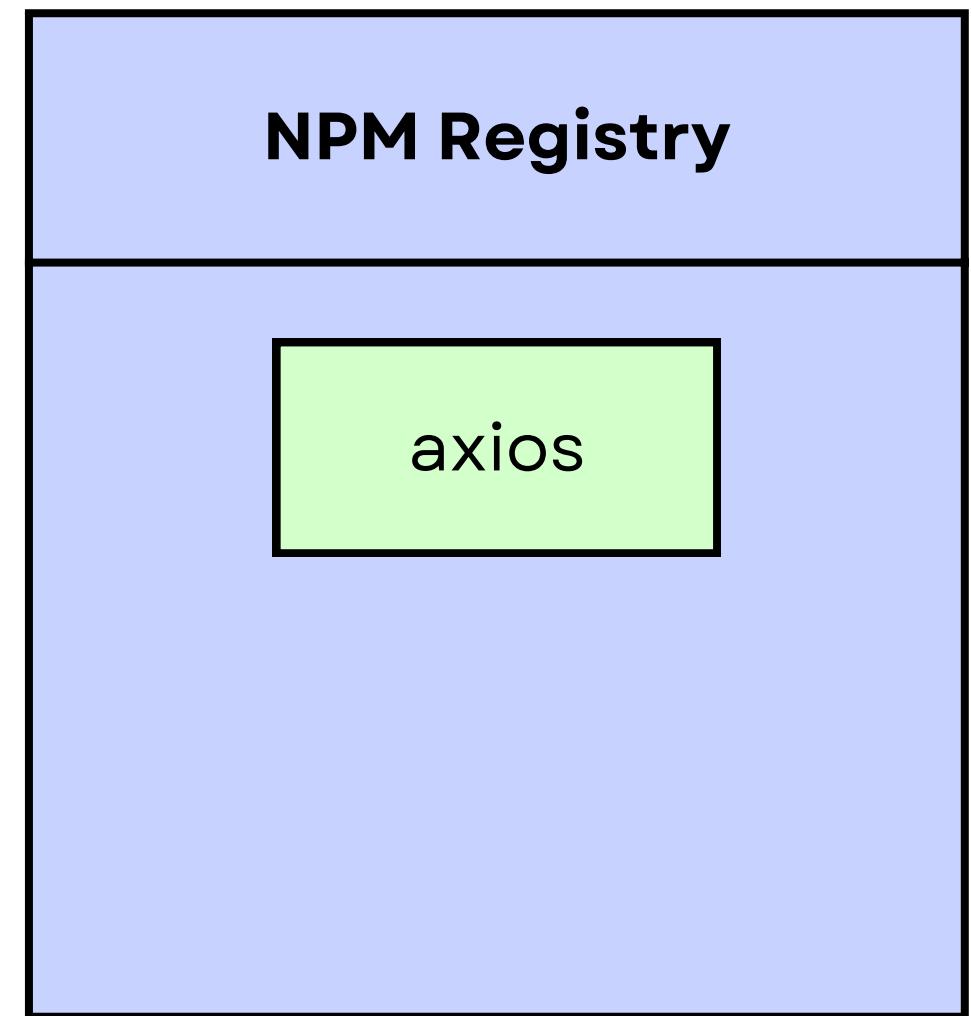
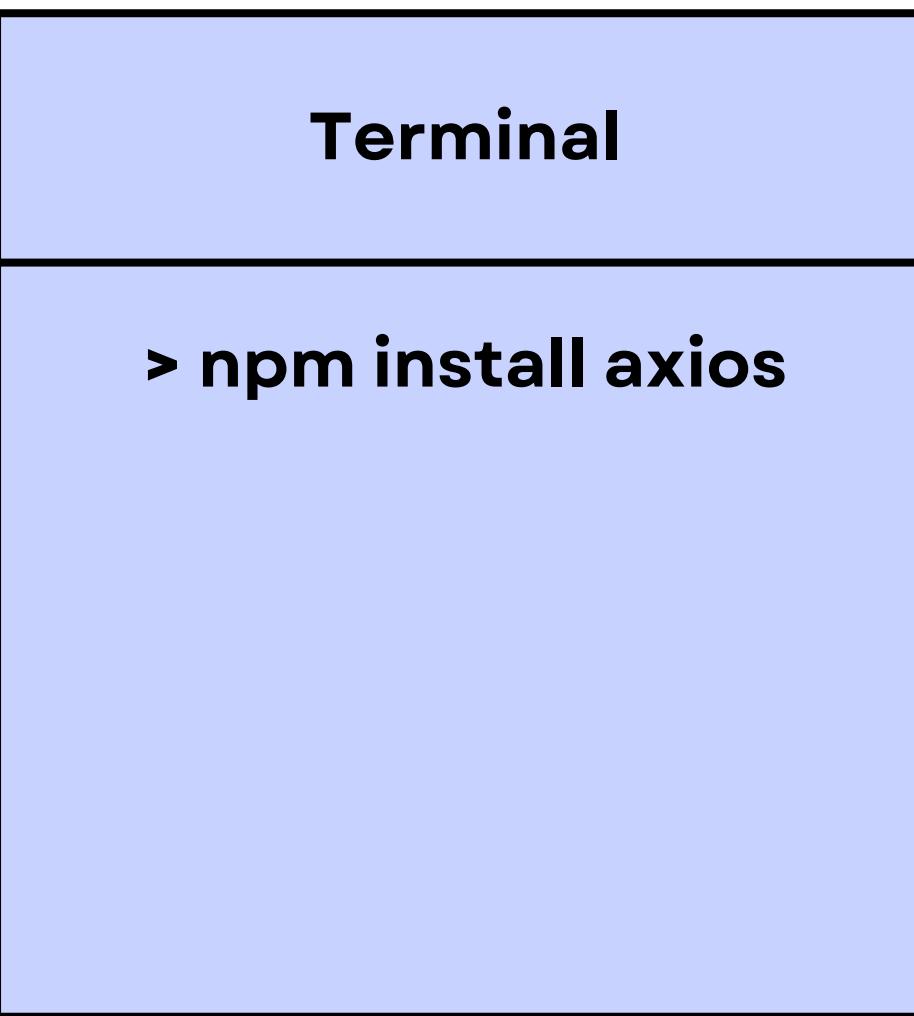
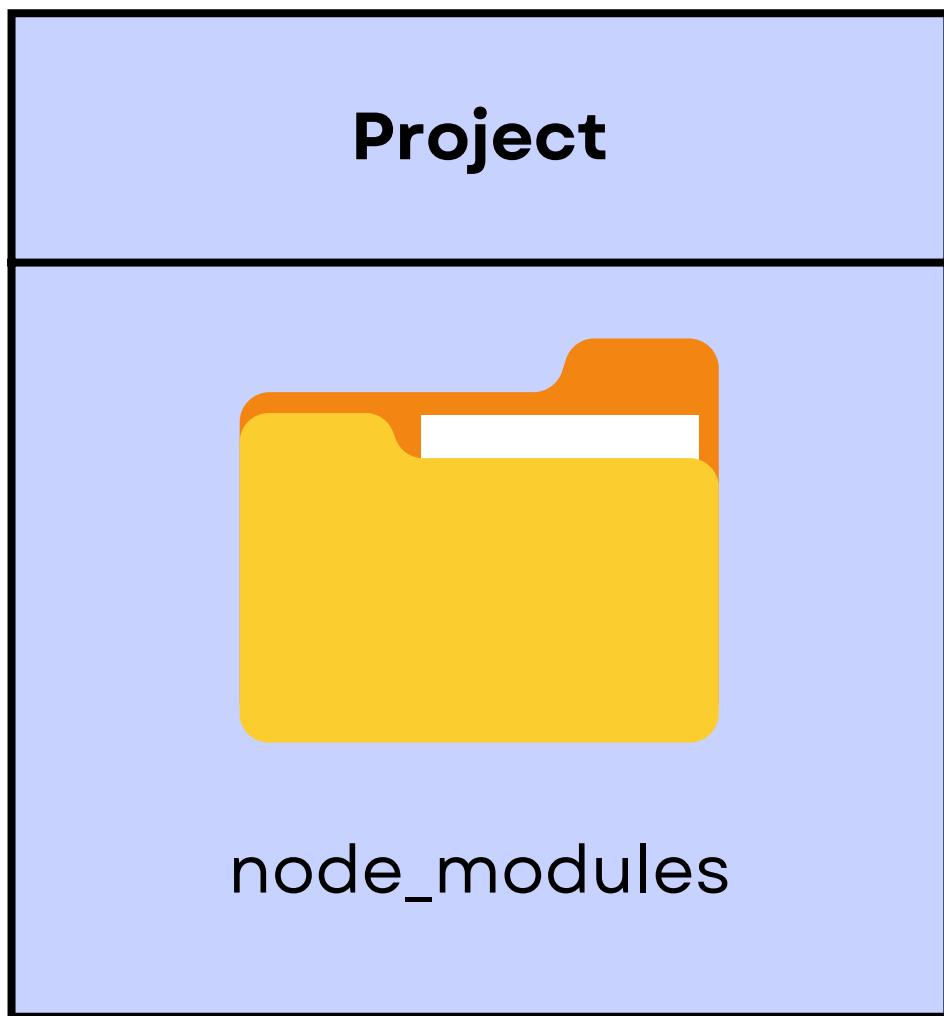
React itself has no functions/tools for making HTTP requests

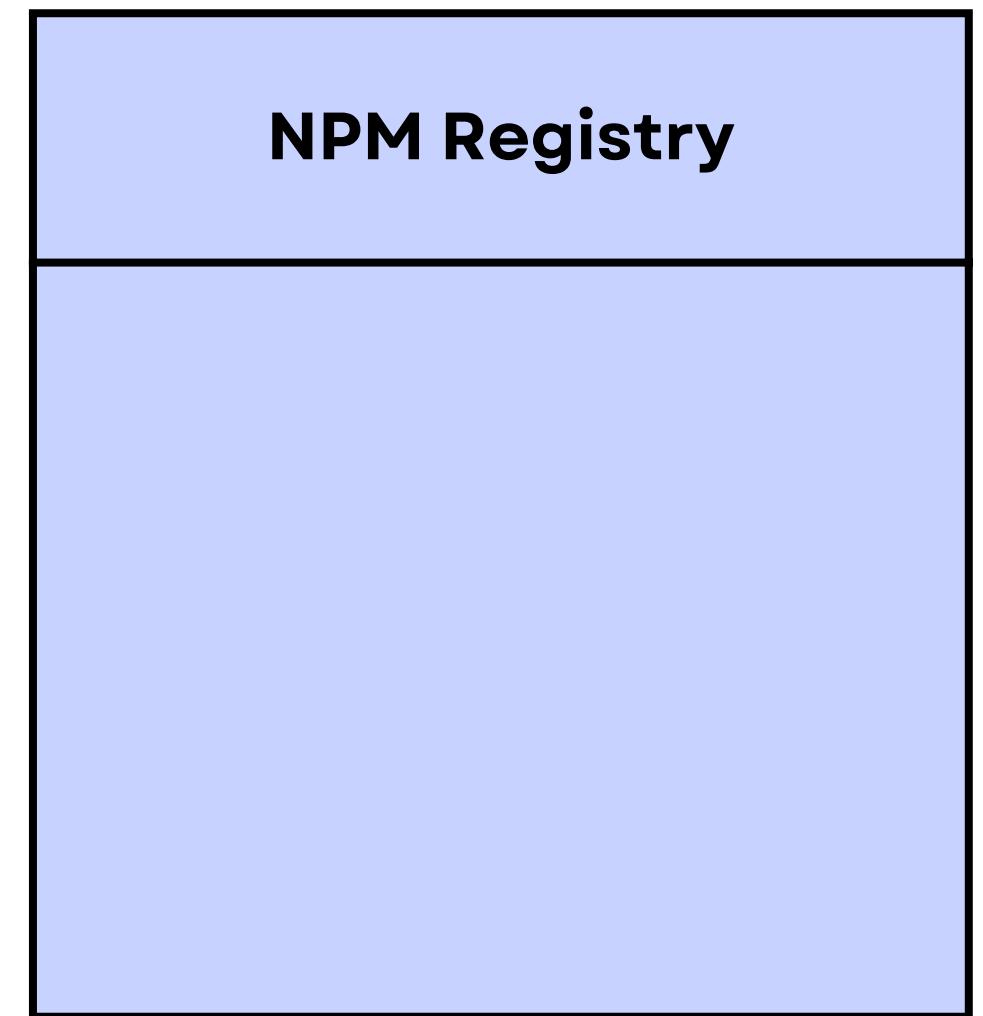
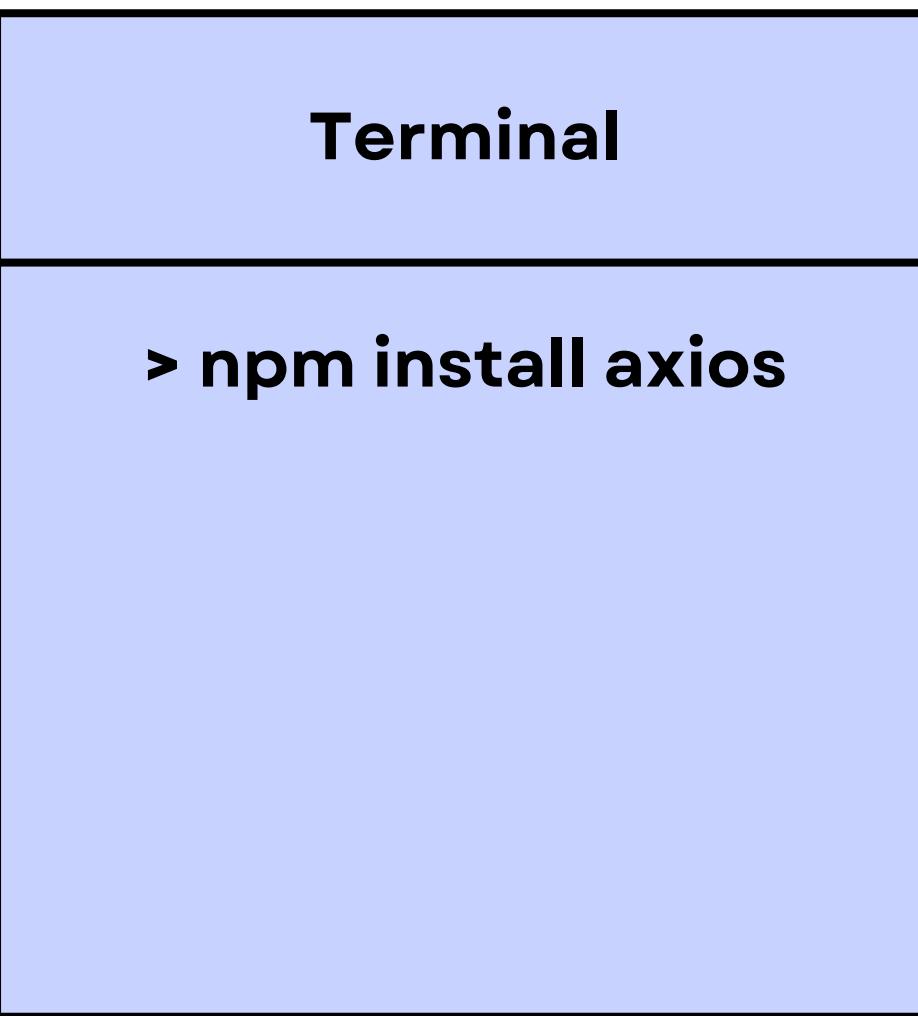
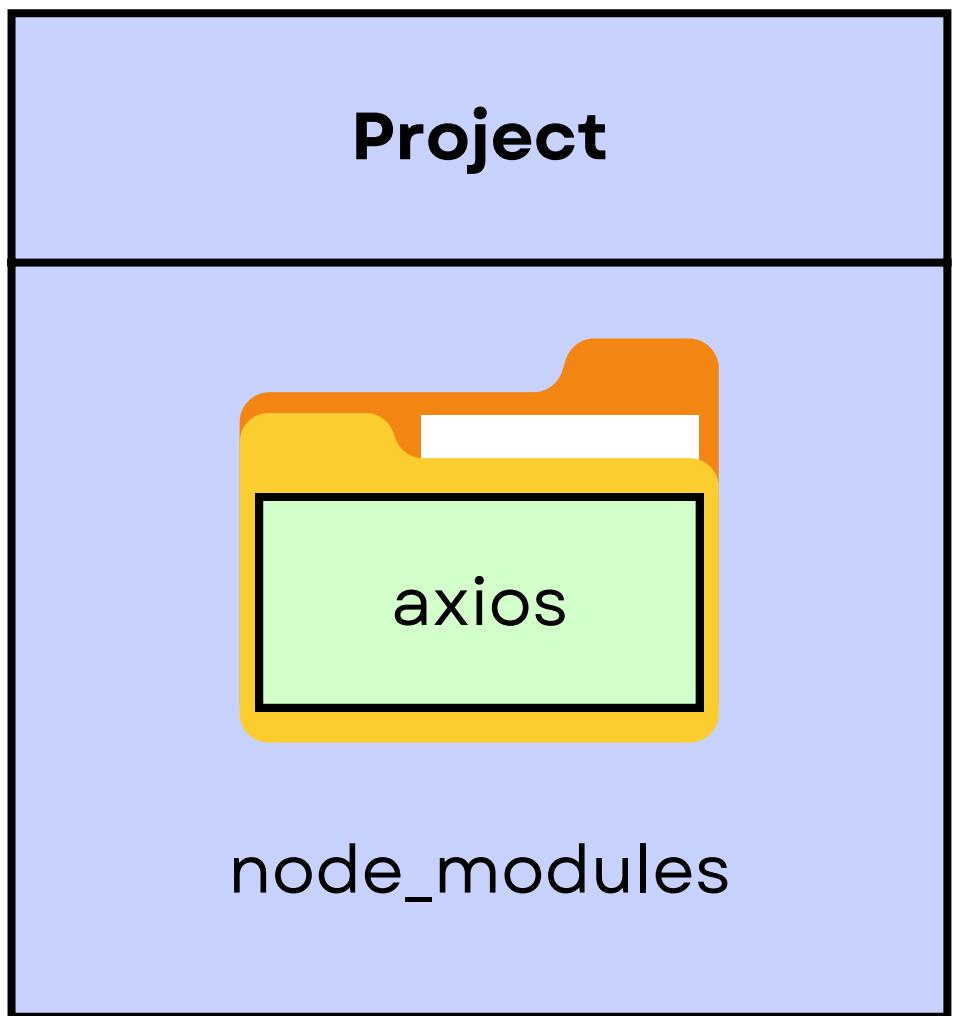


To make requests, we commonly use either Axios or Fetch



In this course we are going to use Axios.  
It is easier to get started with





Request method.  
Can be ‘get’, ‘post’  
‘del’, etc.

Where we want to  
make the request to

```
axios.get(url, {  
  header: {  
    [ ]  
  },  
  params: {  
    [ ]  
  }  
});
```

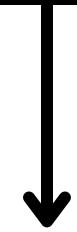
Headers that we want to  
add into the request

Key-value pairs that will be  
turned into a query string  
and added to the URL

## HTTP Request

```
GET http://api.unsplash.com/search/photos?query=oceans
```

```
Authorization: Client-ID kL0FH...
```



```
axios.get('http://api.unsplash.com/search/photos', {  
  header: {  
    Authorization: 'Client-ID kL0FH...etc'  
  },  
  params: {  
    query: 'cars'  
  },  
})
```

JS starts the request...

...then *instantly* runs this line

```
const fetchData = () => {  
  const response = makeRequest();  
  
  console.log(response);  
}
```

### Probably not good!

JS doesn't automatically pause when you make a request  
Response hasn't been received yet!

*Time*



User enters a search term and presses enter

Make a request to the API

*Times passes...*

Get a response!

Use the response in some way

JS starts the request...

...then *instantly* runs this line

```
const fetchData = () => {  
  const response = makeRequest();  
  
  console.log(response);  
}
```

### Probably not good!

JS doesn't automatically pause when you make a request  
Response hasn't been received yet!



**Probably need something like this.**  
Don't try to work with the response until it has  
actually been received.



**Probably need something like this.**  
Don't try to work with the response until it has actually been received.

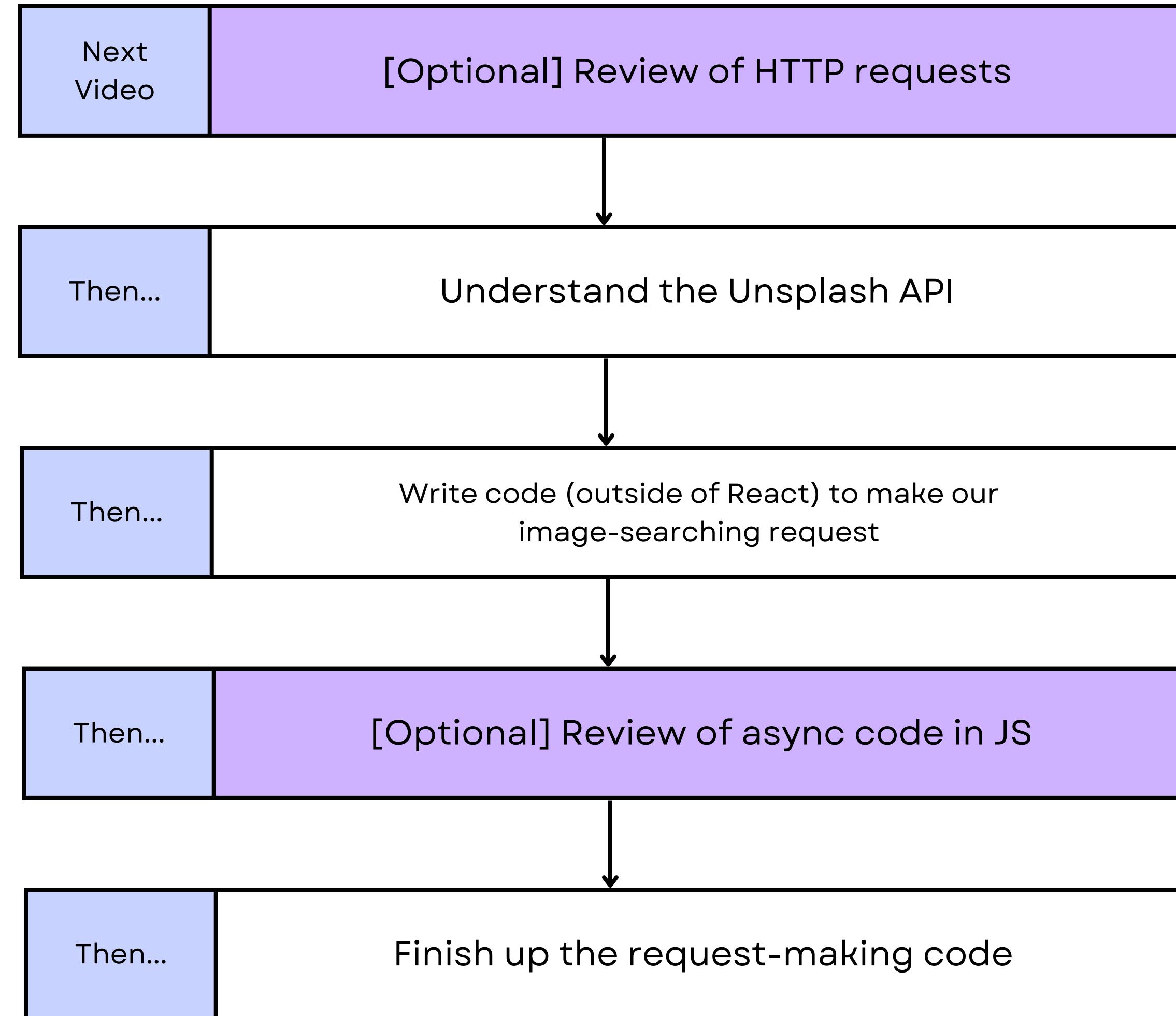
JS starts the request...

Tell JS to wait for a response

Response has been received and can be the read

```
const fetchData = async () => {  
  const response = await axios.get('...');  
  
  console.log(response);  
}
```

**Use `async/await` to tell JS to wait for the request to finish before moving on**



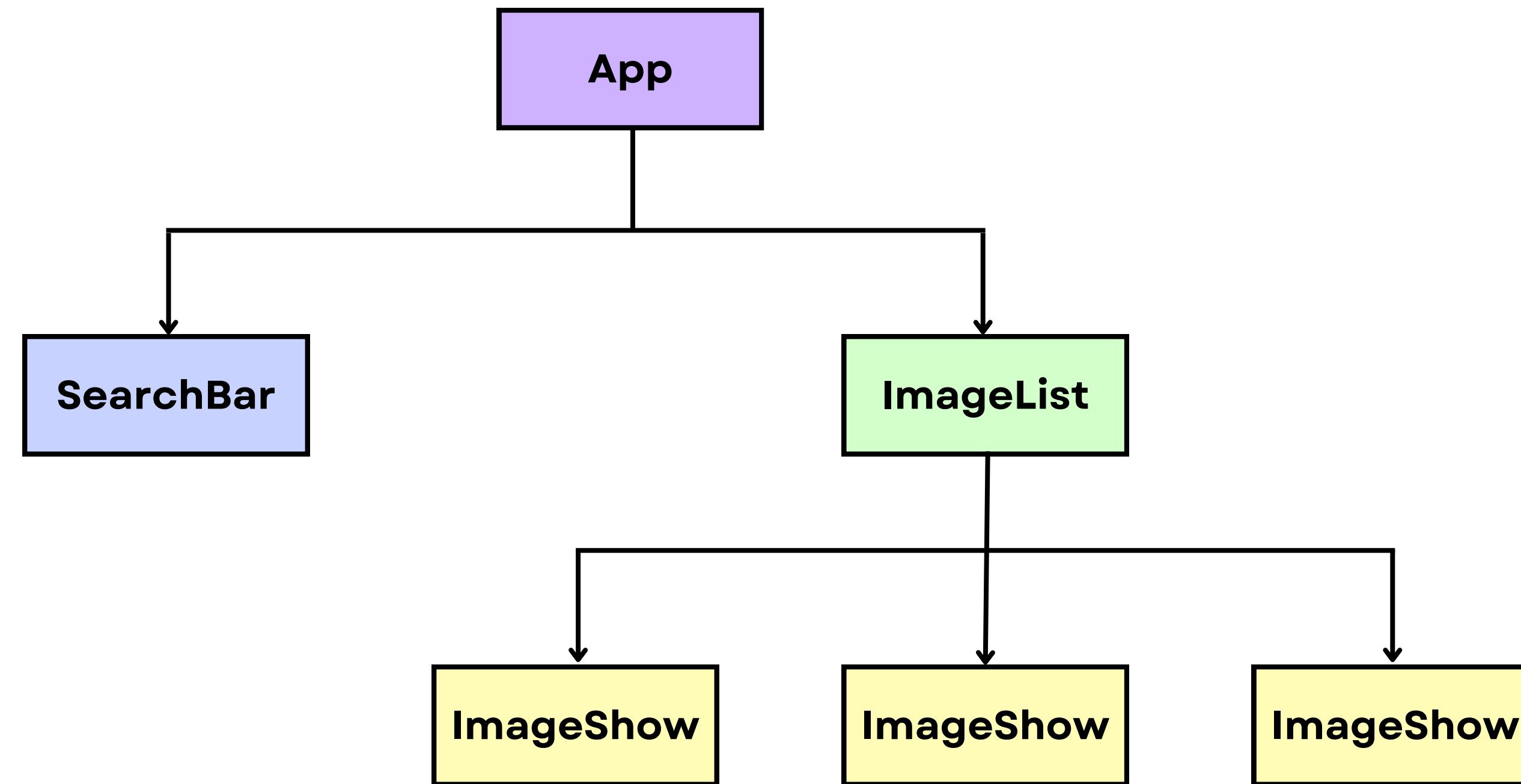
How does data flow through a React app?



Where do we do data fetching?

Where do we define state?

How do we share info between components?



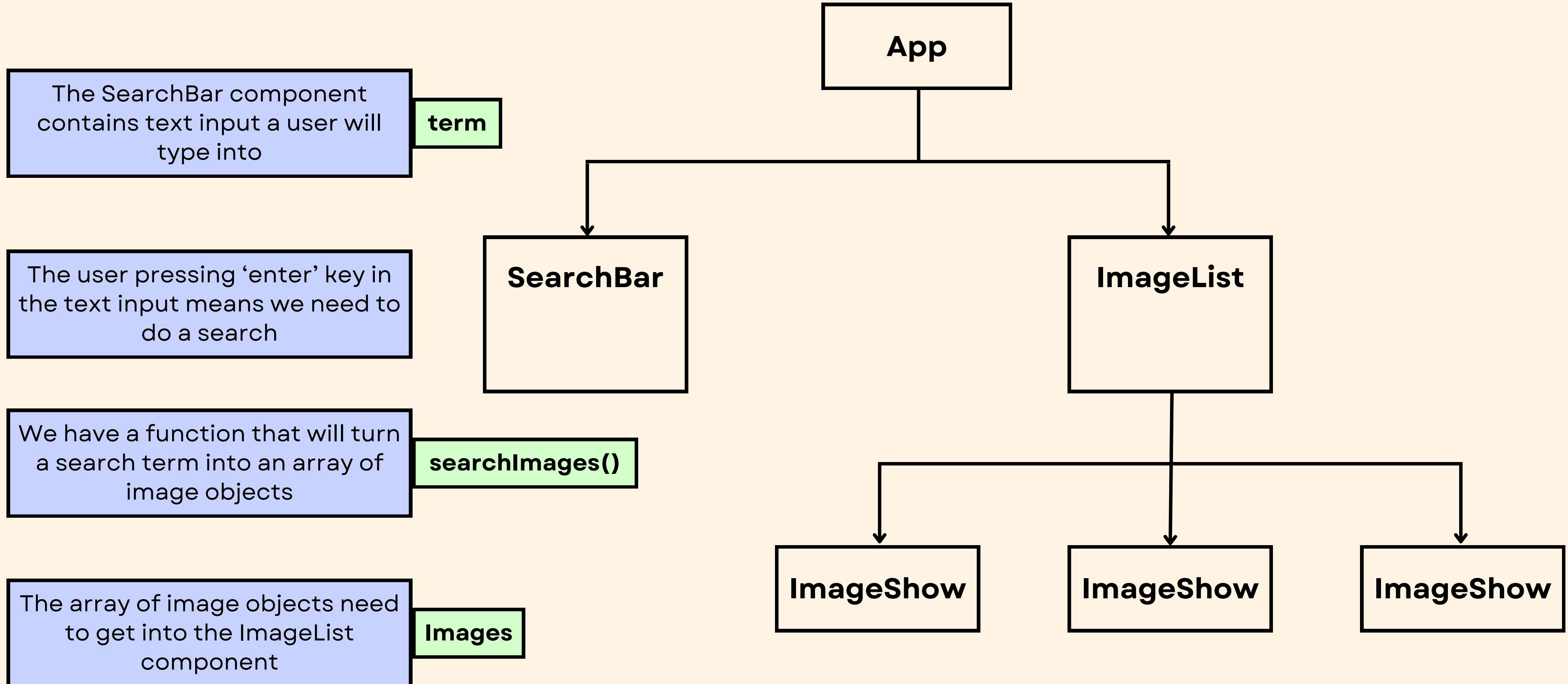
# Facts of This App

The SearchBar component contains the text input a user will type into

The user pressing ‘enter’ key in the text input means we need to do a search

We have a function that will turn a search them into an array of image objects

The array of image objects need to get into the ImageList component

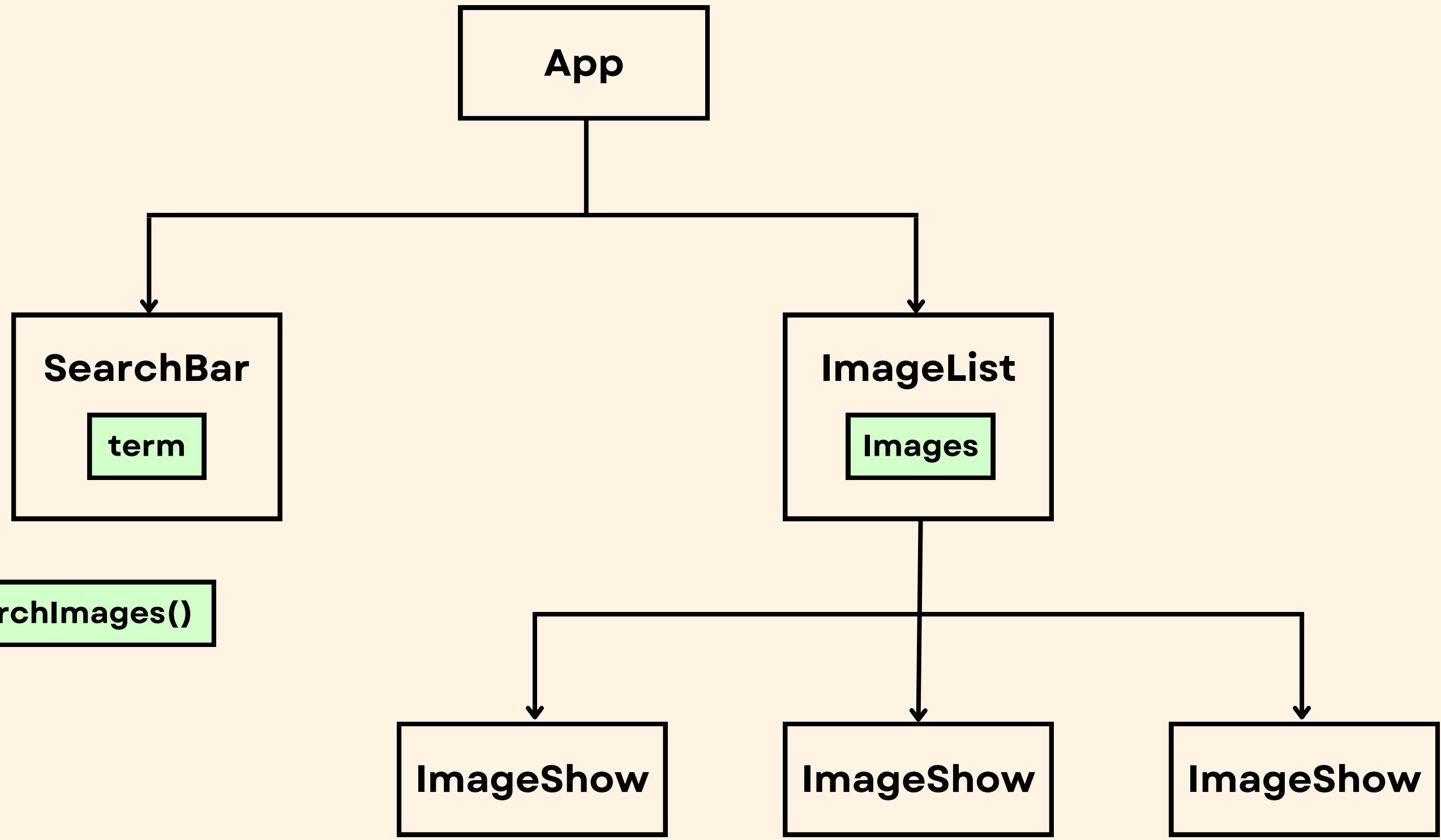


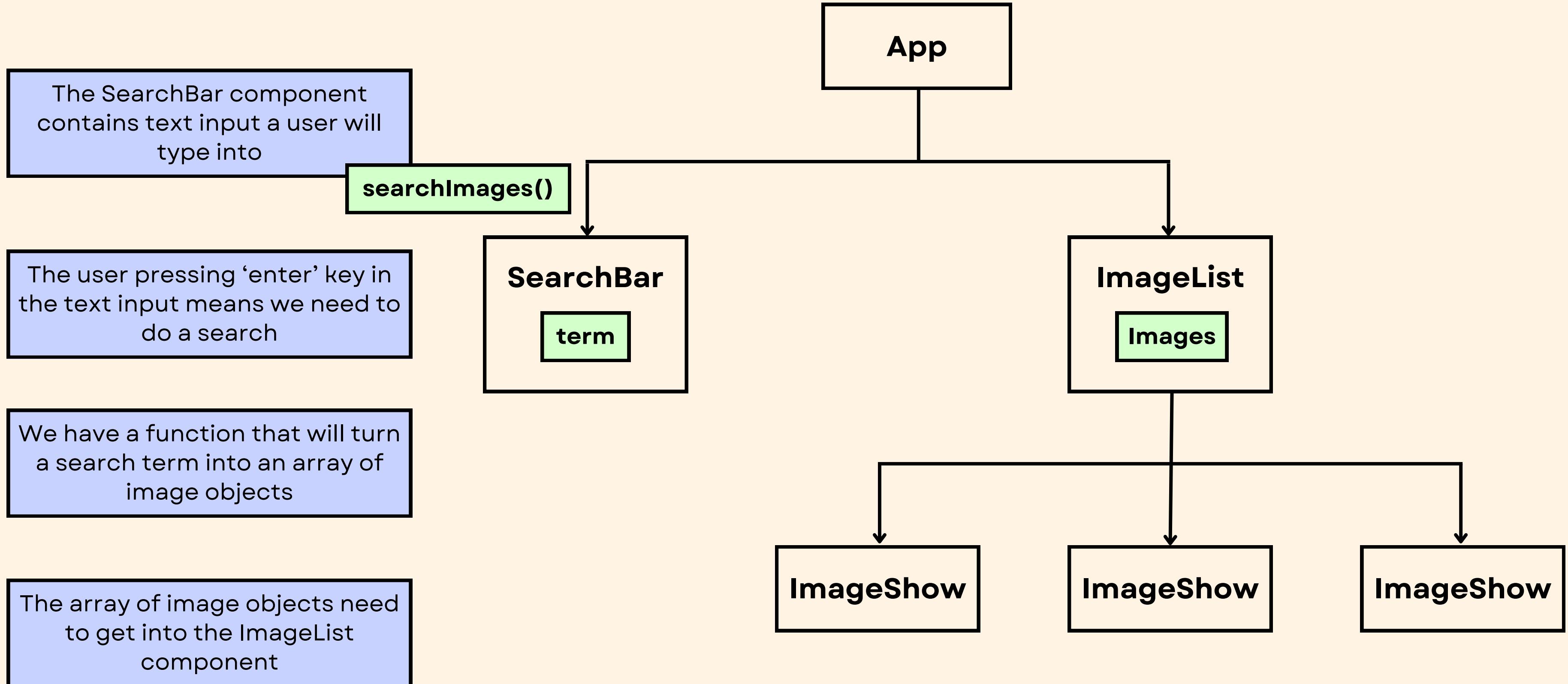
The SearchBar component contains text input a user will type into

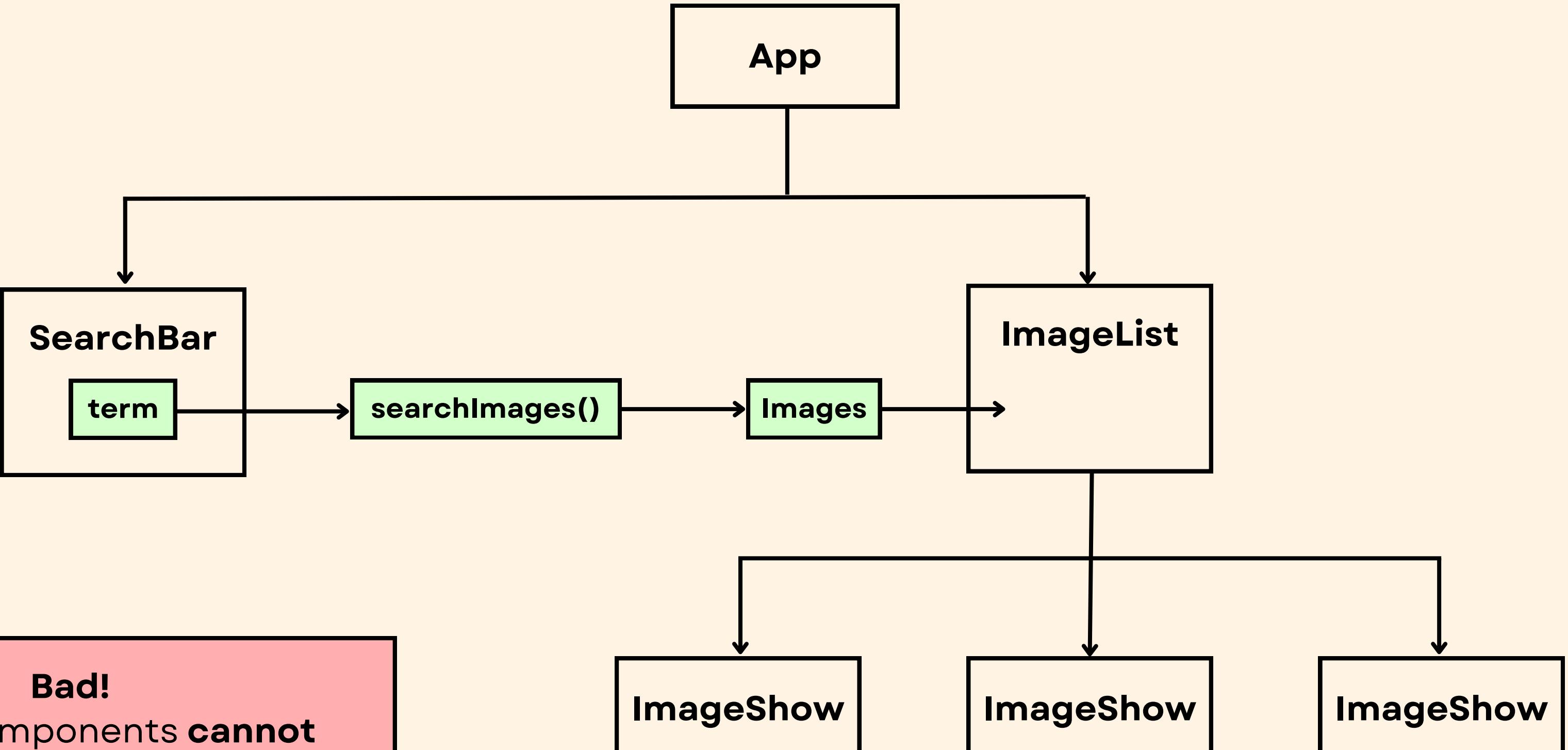
The user pressing 'enter' key in the text input means we need to do a search

We have a function that will turn a search term into an array of image objects

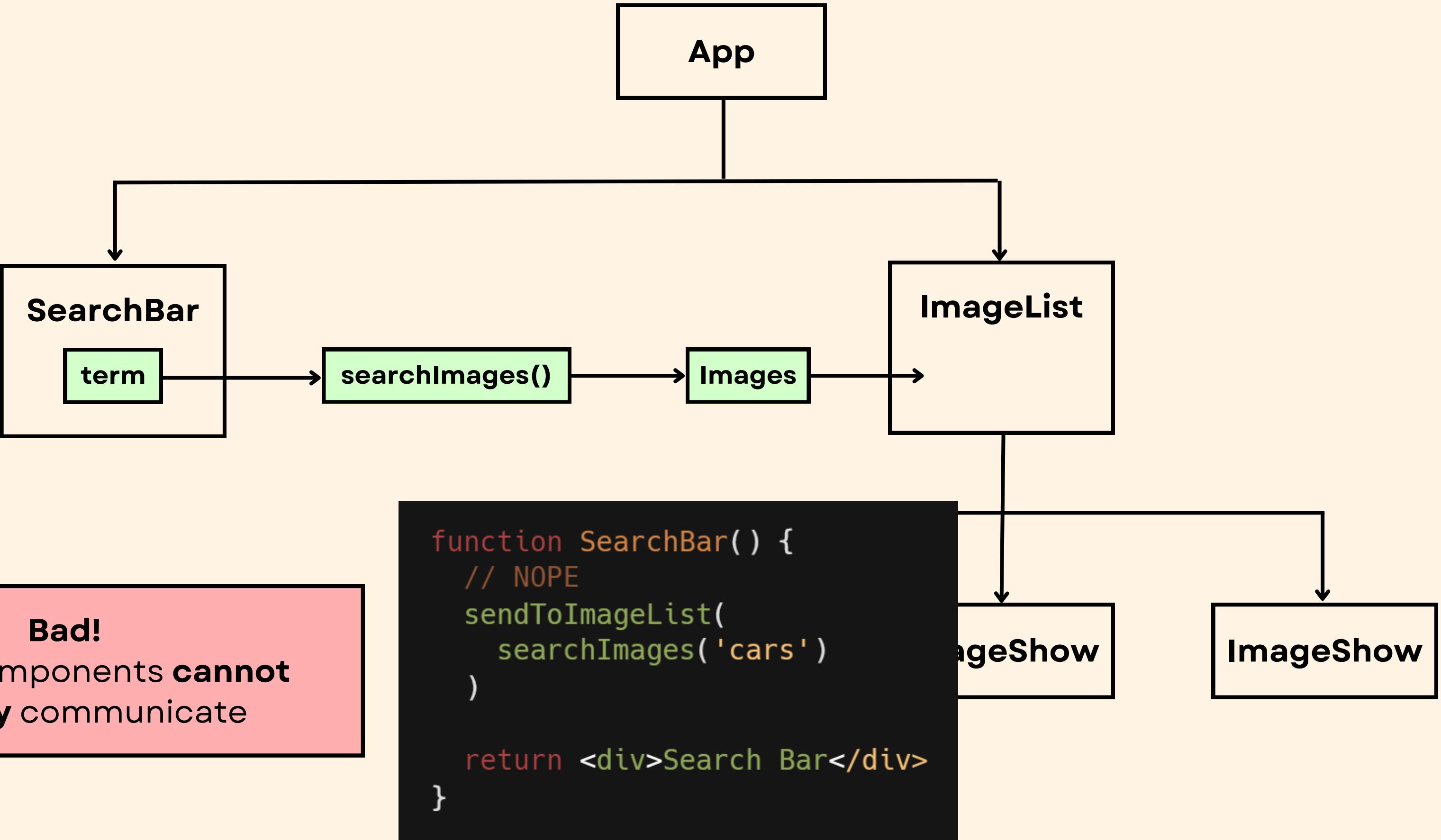
The array of image objects need to get into the ImageList component

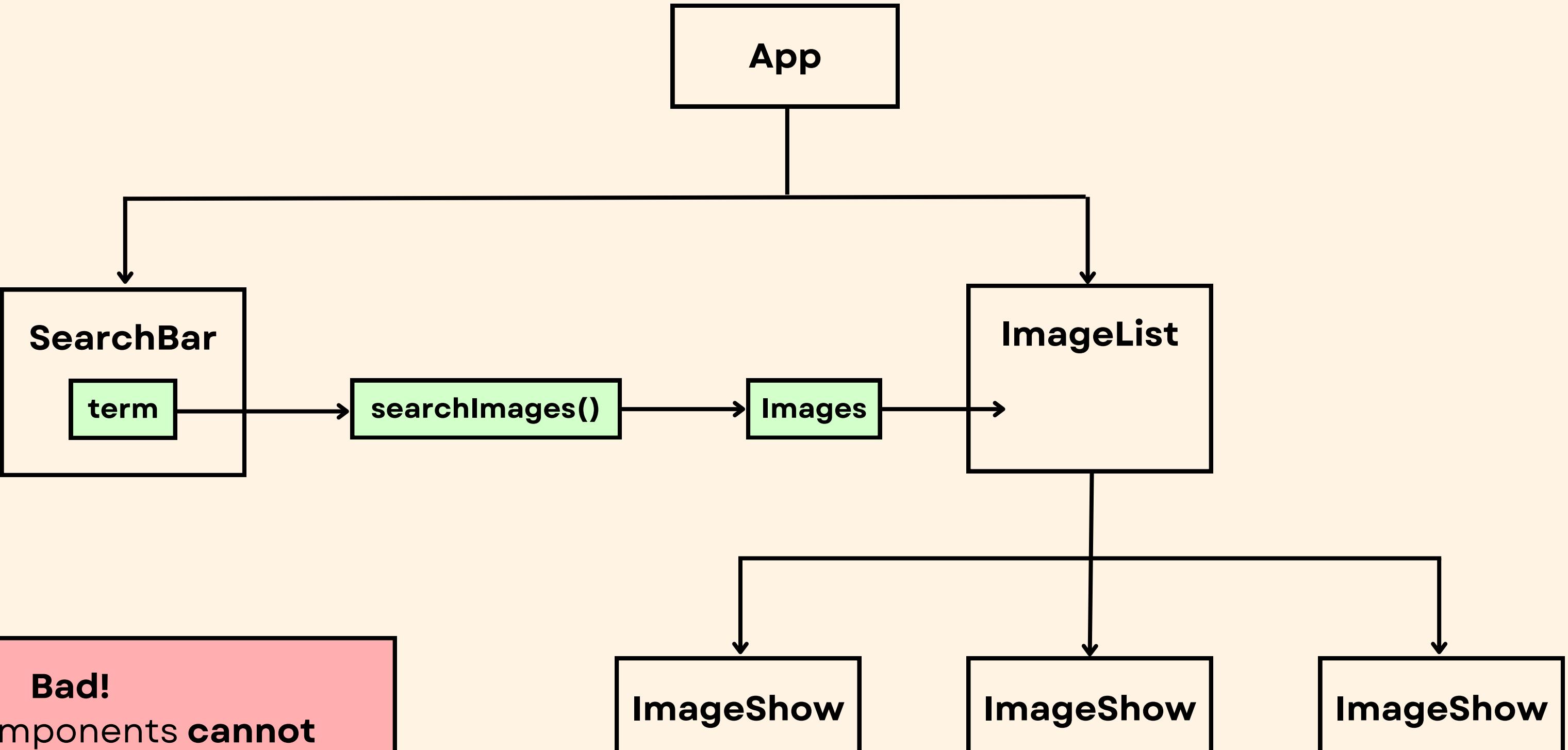






**Bad!**  
Sibling components **cannot**  
**directly** communicate

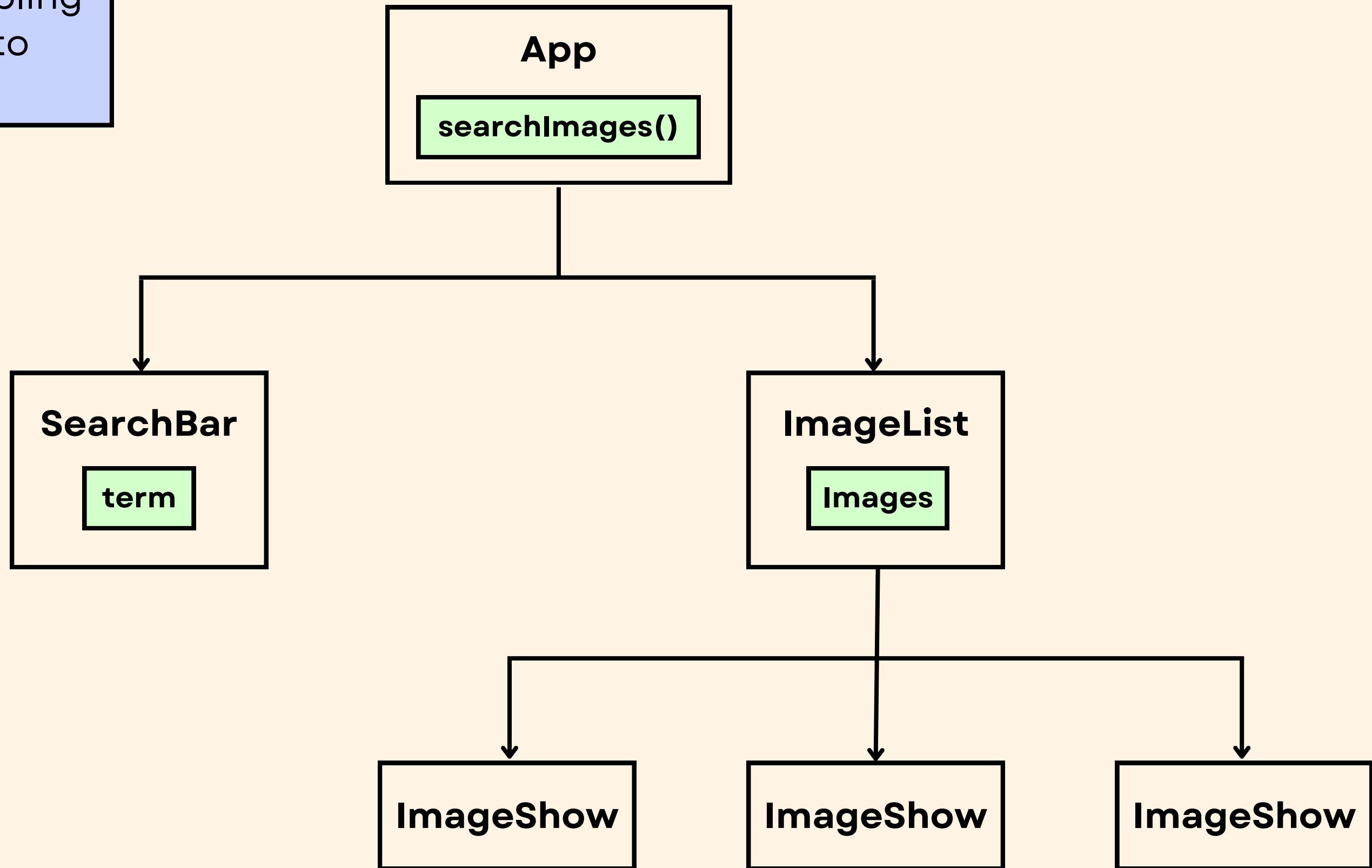




**Bad!**  
Sibling components **cannot**  
**directly** communicate

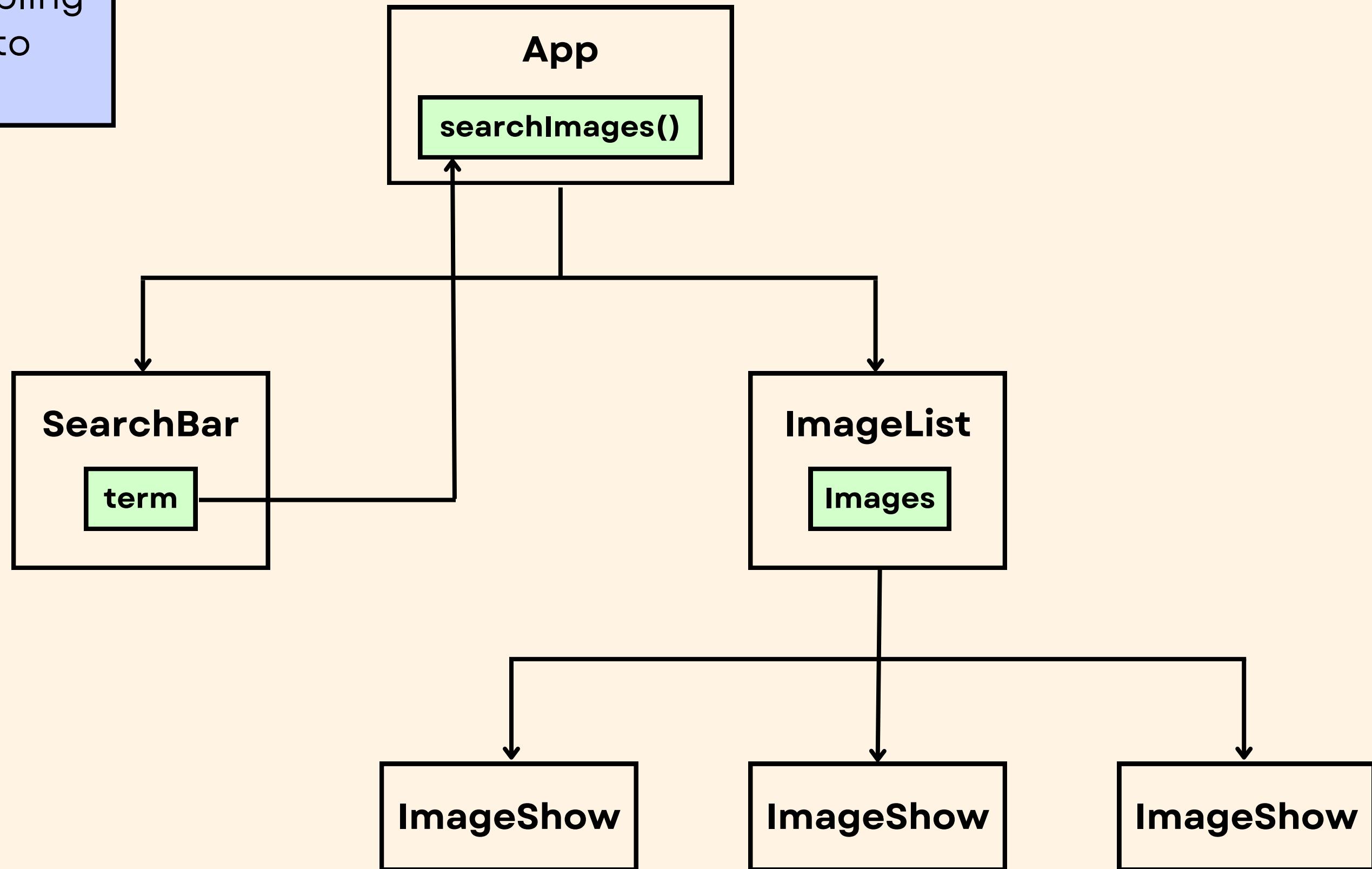
**Good!**

To share info between sibling component, we have to involve the parent



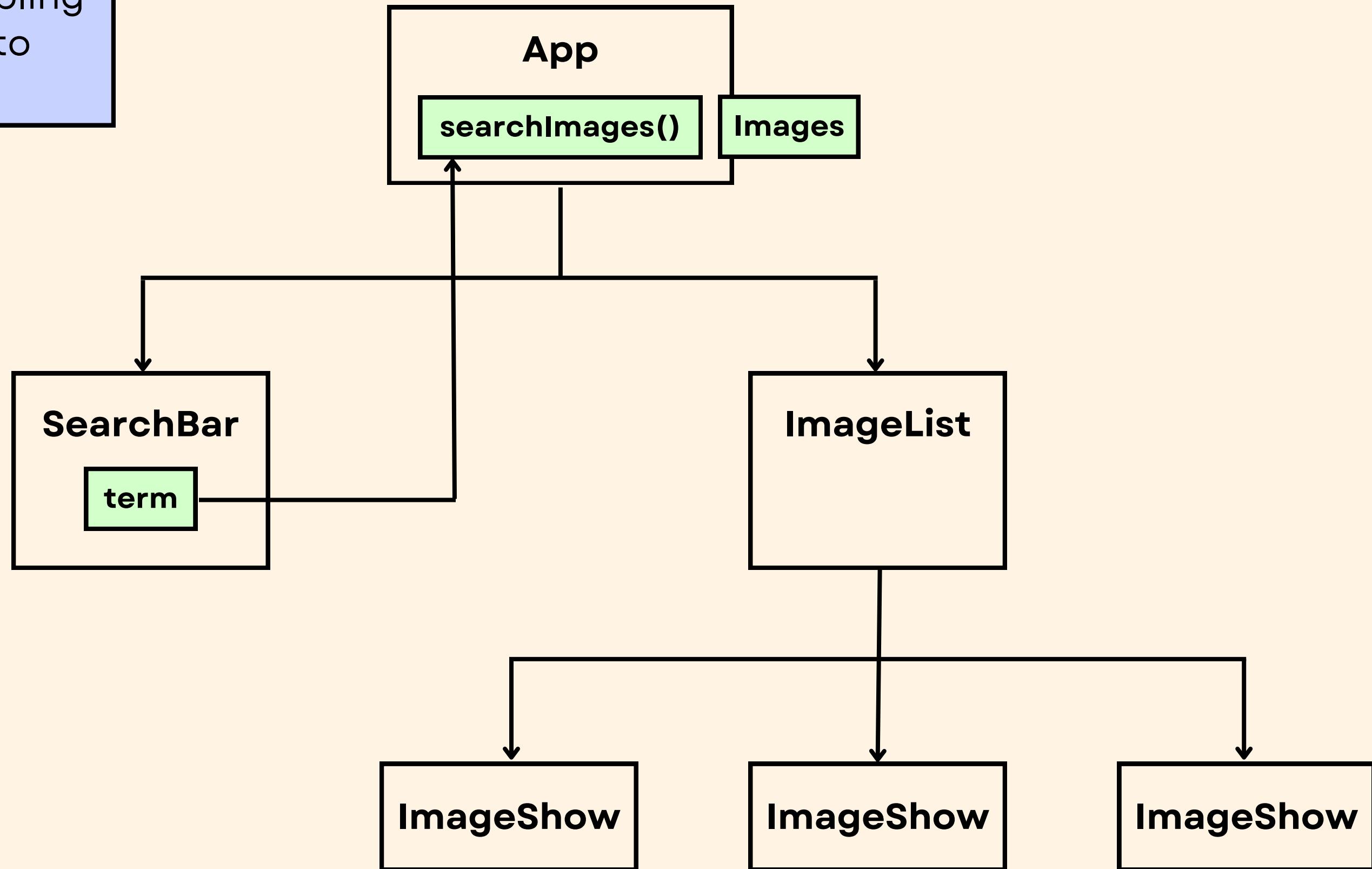
**Good!**

To share info between sibling component, we have to involve the parent



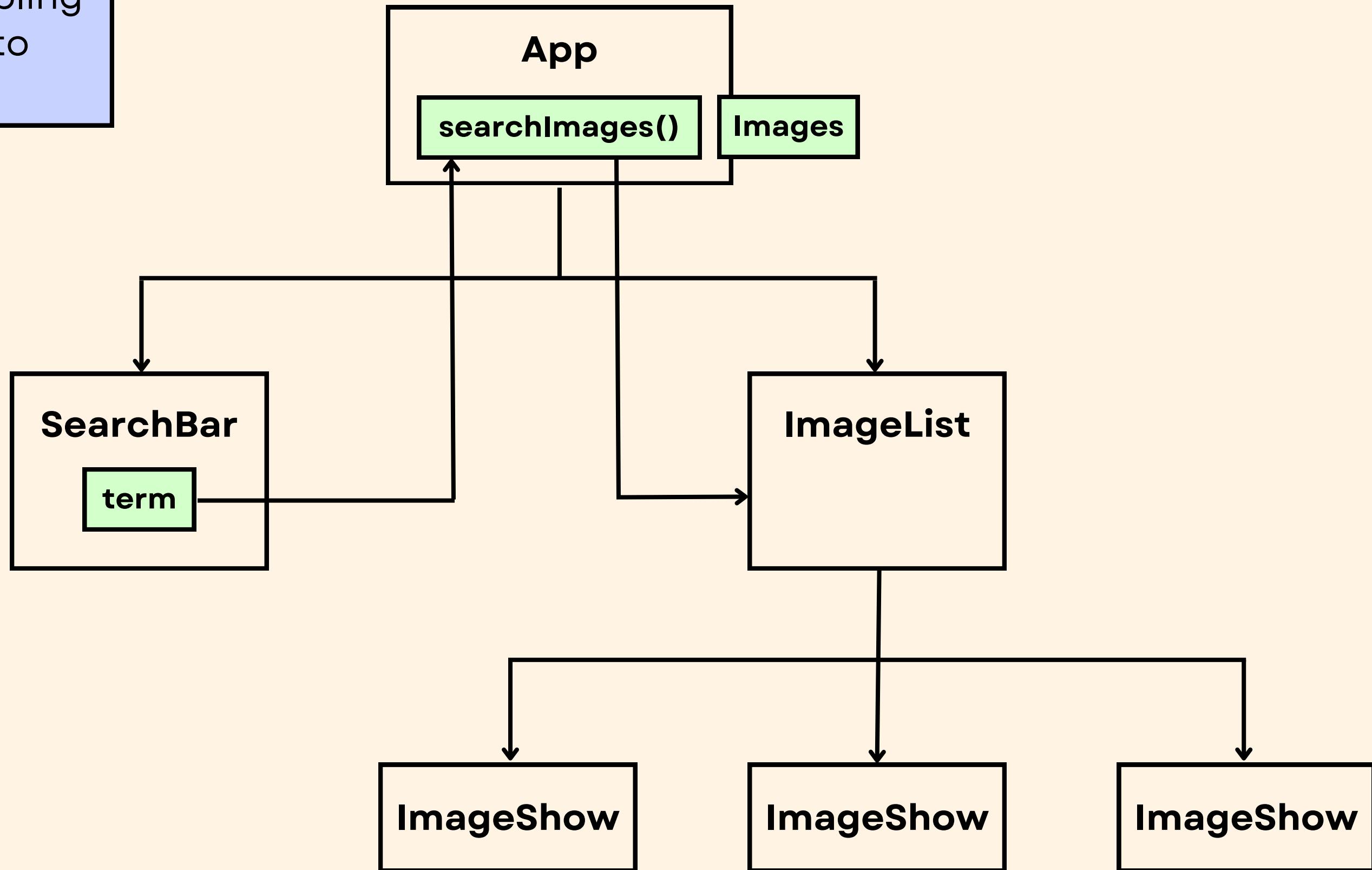
**Good!**

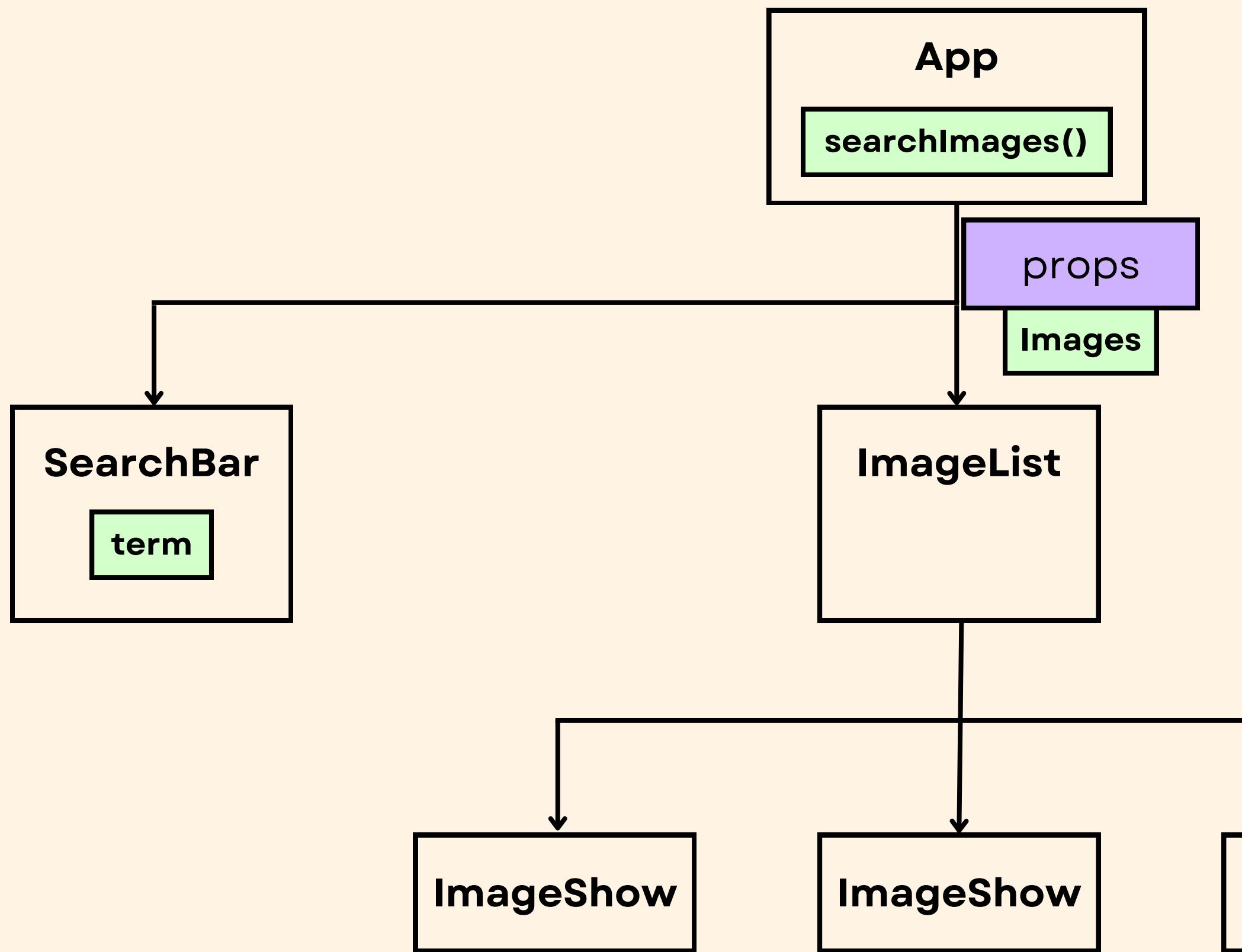
To share info between sibling component, we have to involve the parent



**Good!**

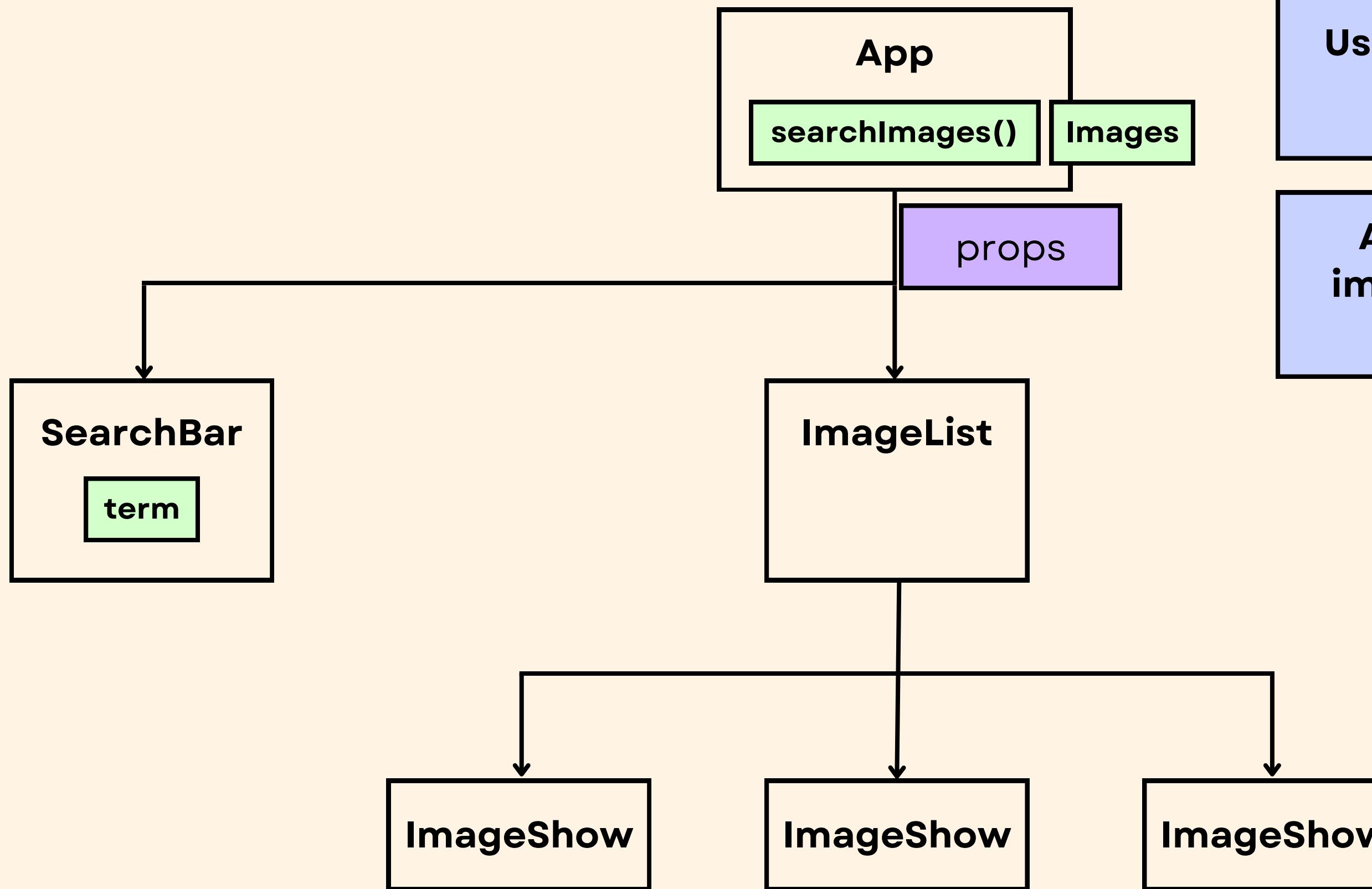
To share info between sibling component, we have to involve the parent





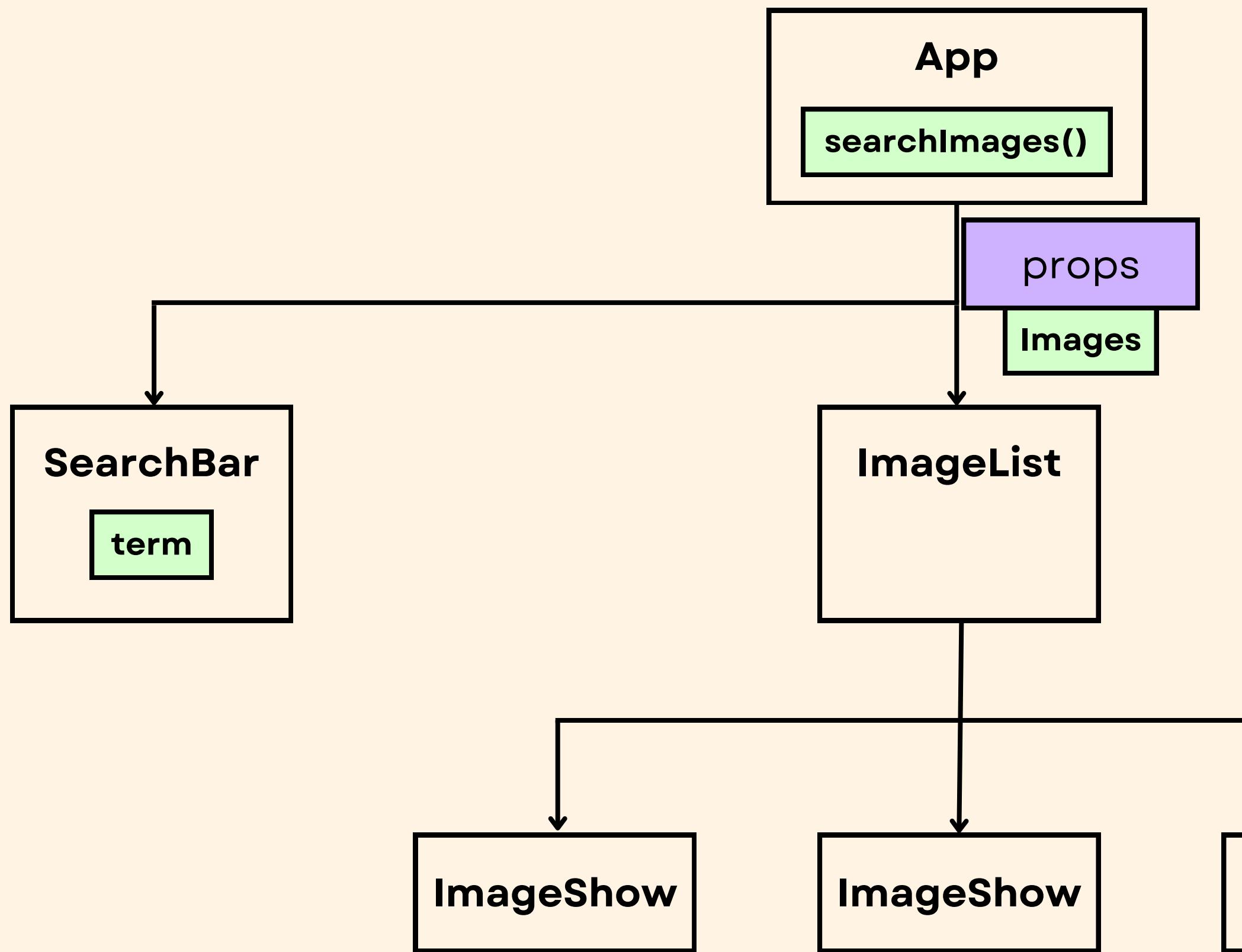
**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**



**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**



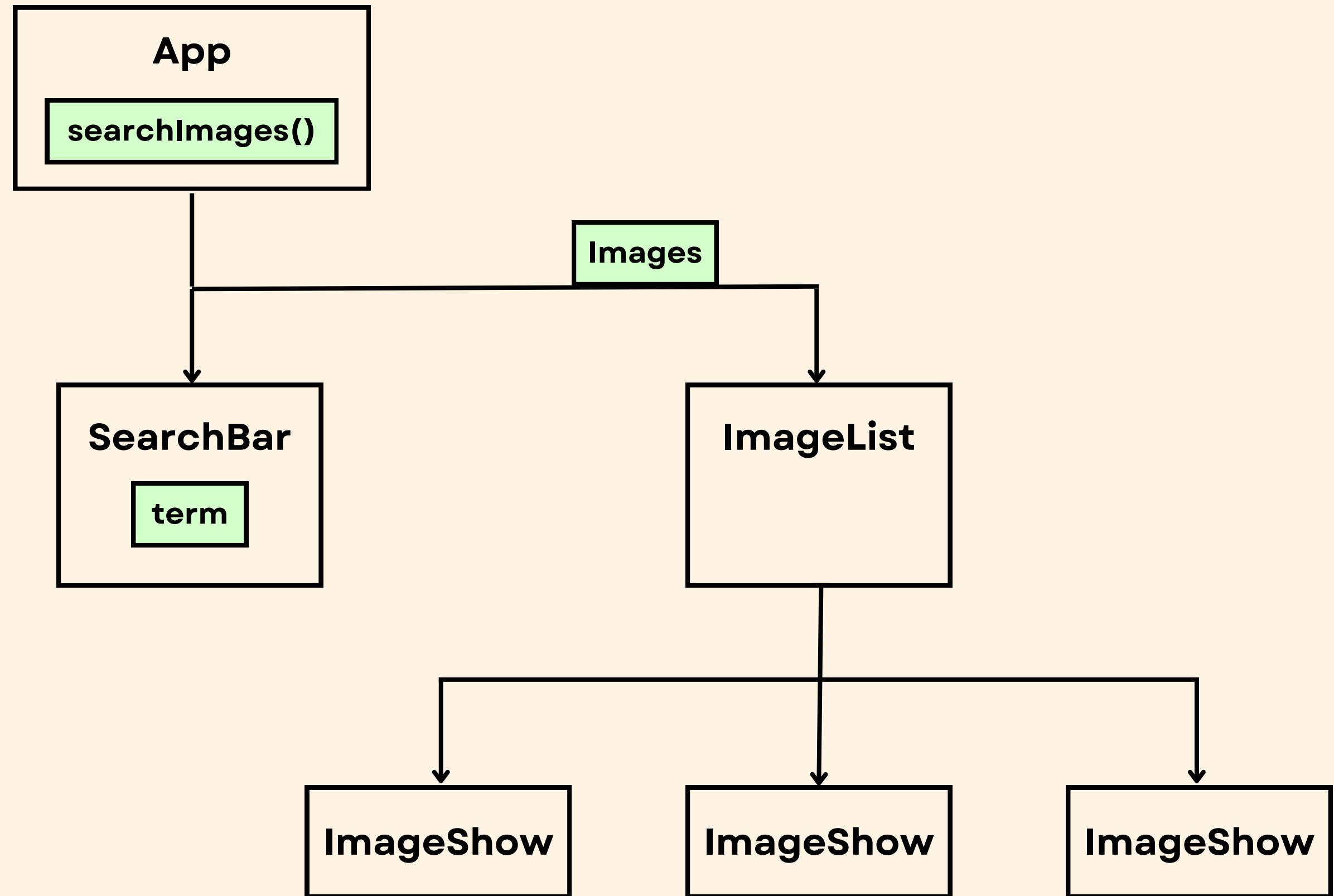
**Use props to communicate from parent to child**

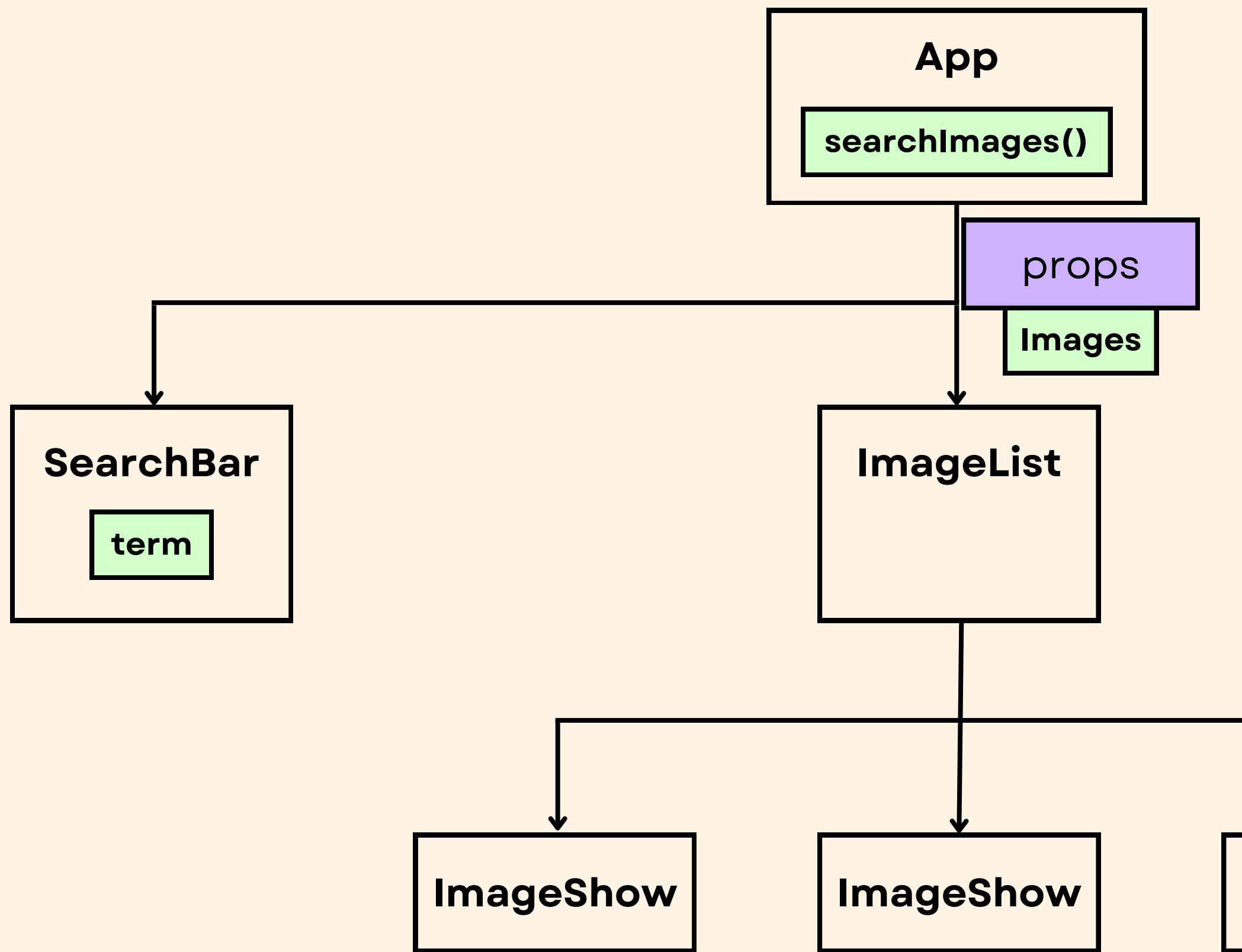
**App can send the list of images down to ImageList using props!**

**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

**????**





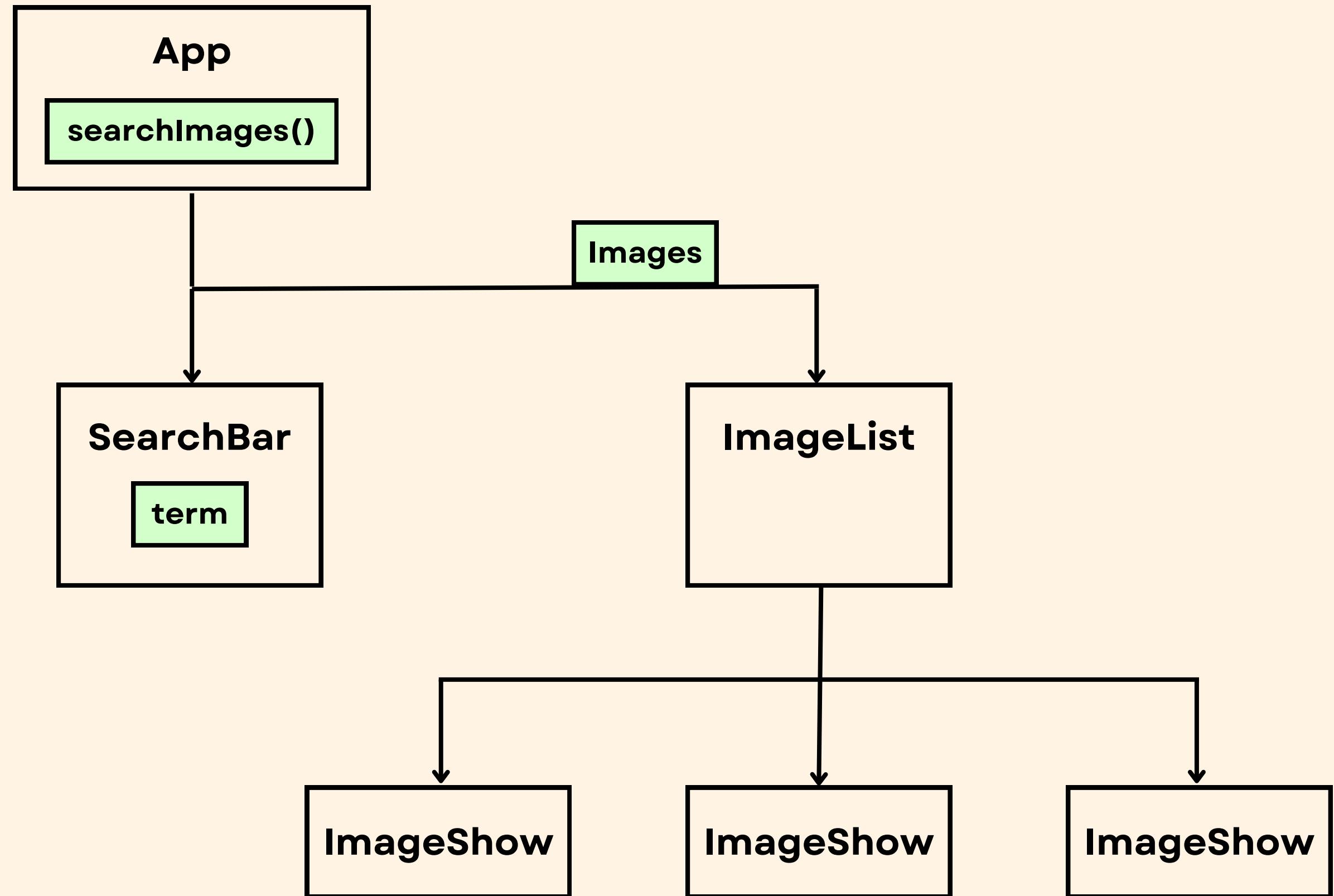
**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**

**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

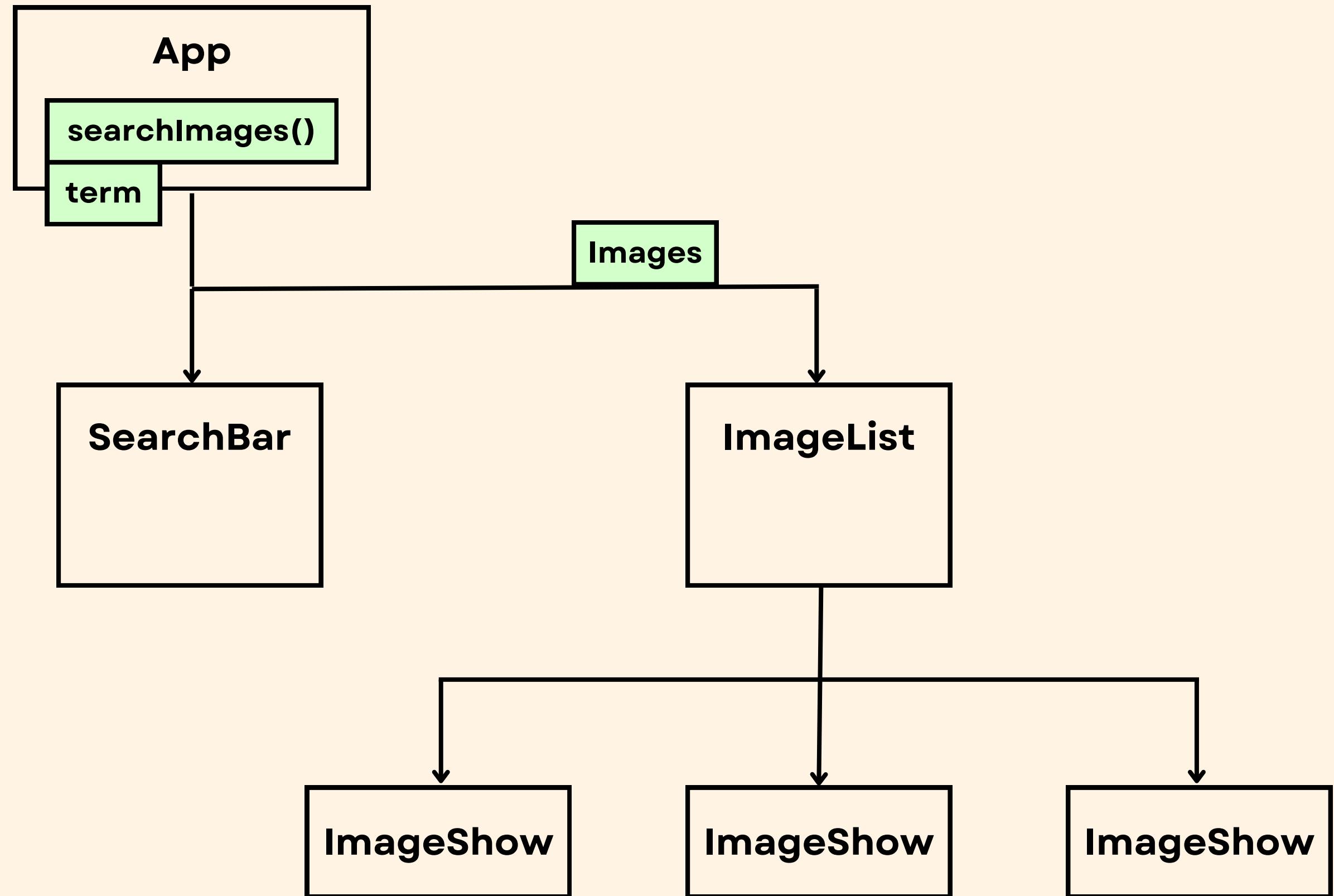
**????**



**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

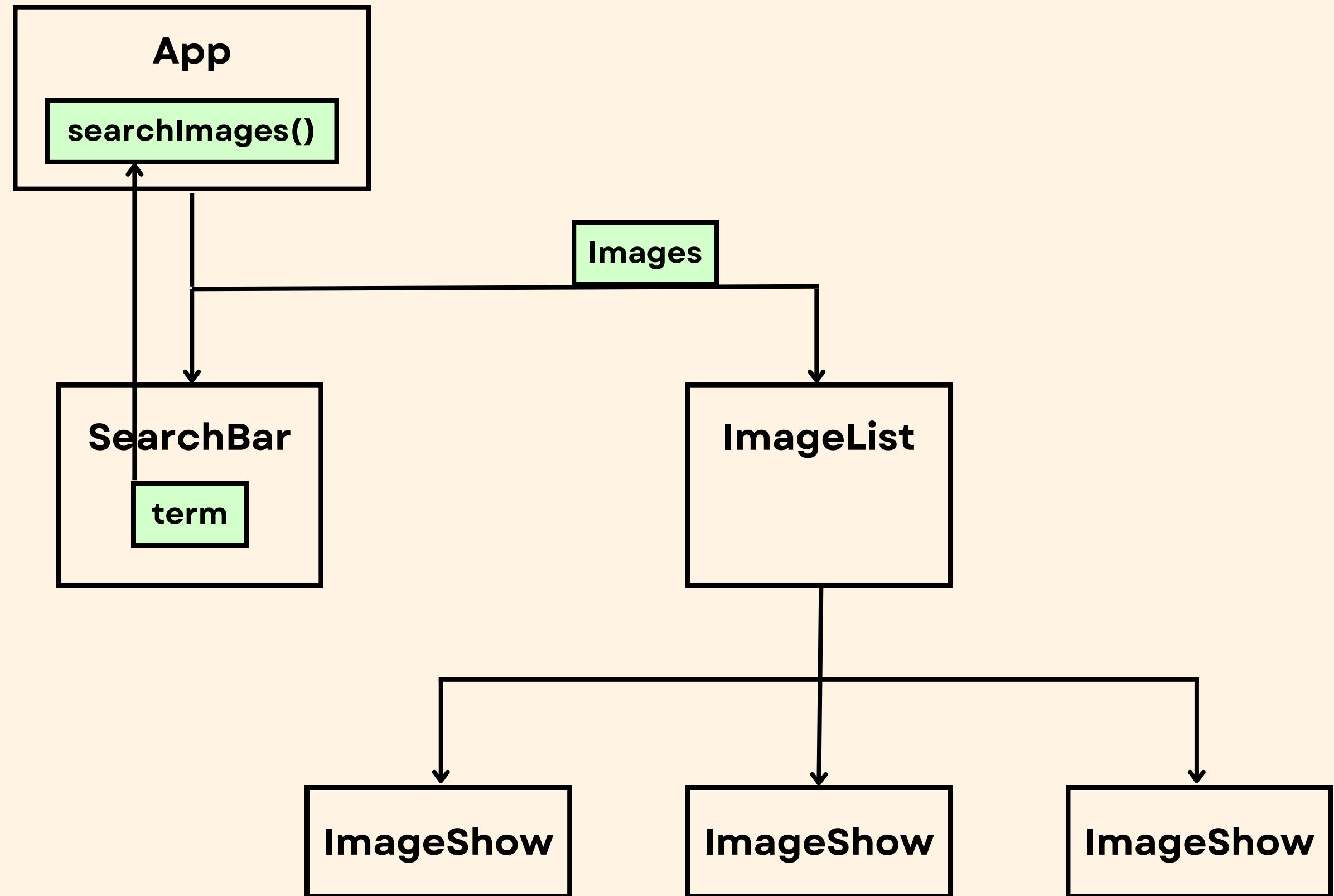
**????**



**When the user presses the  
'enter' key we need to get  
this search term up to the App**

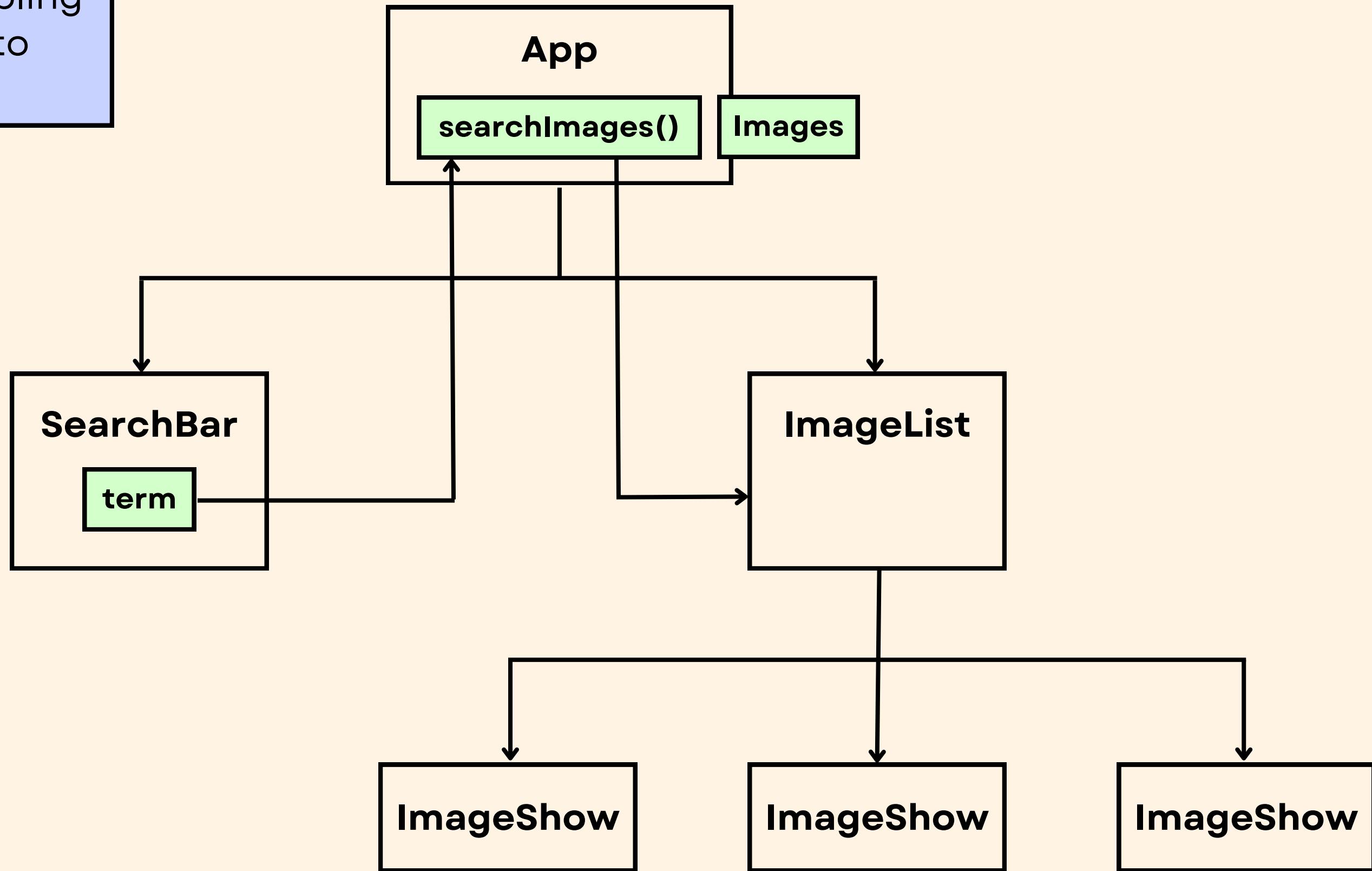
**Child to parent  
communication**

**????**



**Good!**

To share info between sibling component, we have to involve the parent



How does data flow through a React app?



Where do we do data fetching?

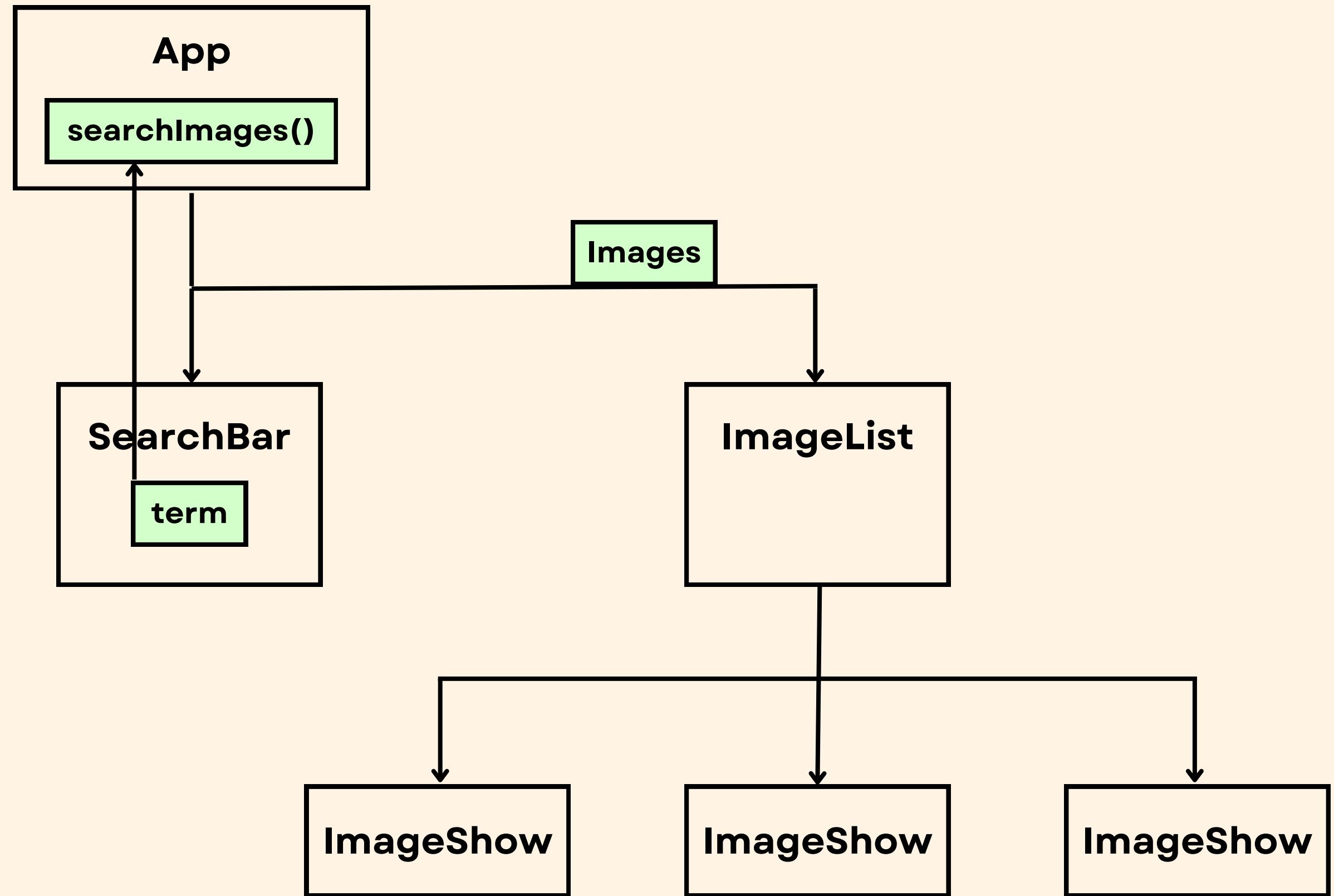
Where do we define state?

How do we share info between components?

**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

**????**



**Parent**



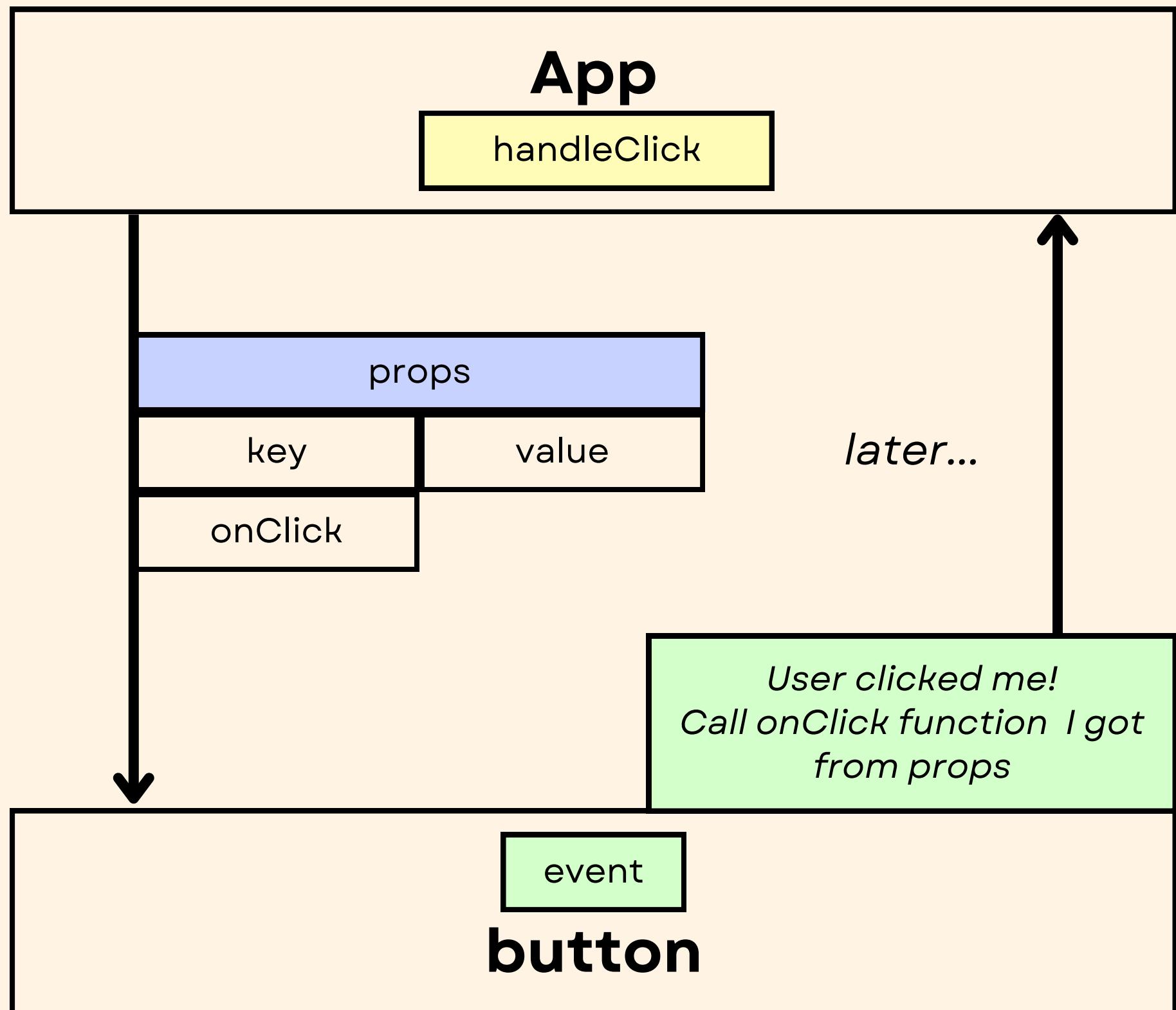
props	
key	value
type	“cat”

**Props**  
Communicate from  
Parent **down** to Child

Many blog posts + docs (even me!) say props is  
**only for parent to child communication**



**Not super accurate**

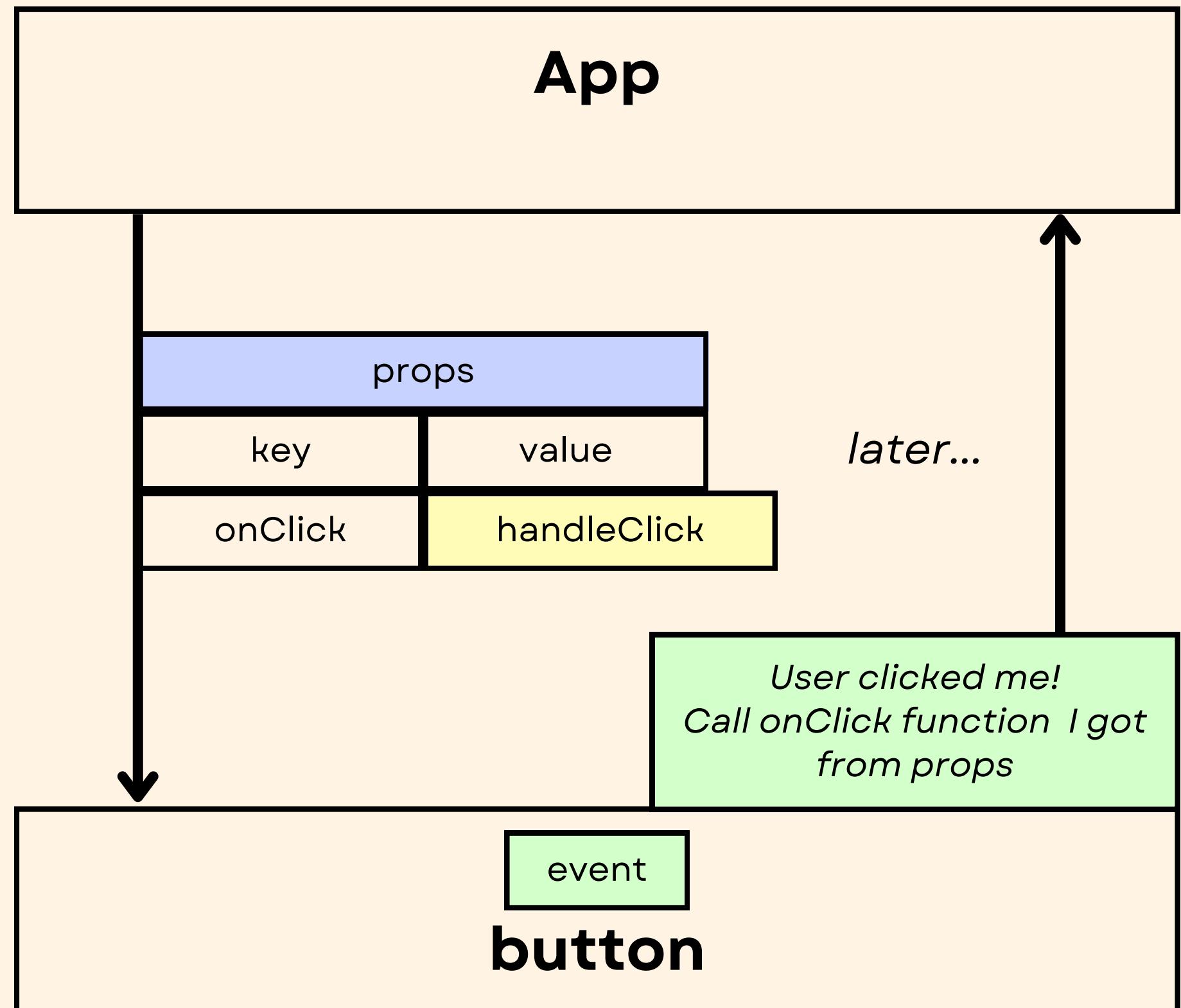


Normal Event Handler

Pass event handler down  
using the *props* system

Handler gets called with an  
event object

**Child to Parent**  
communication, kinda  
using *props*?

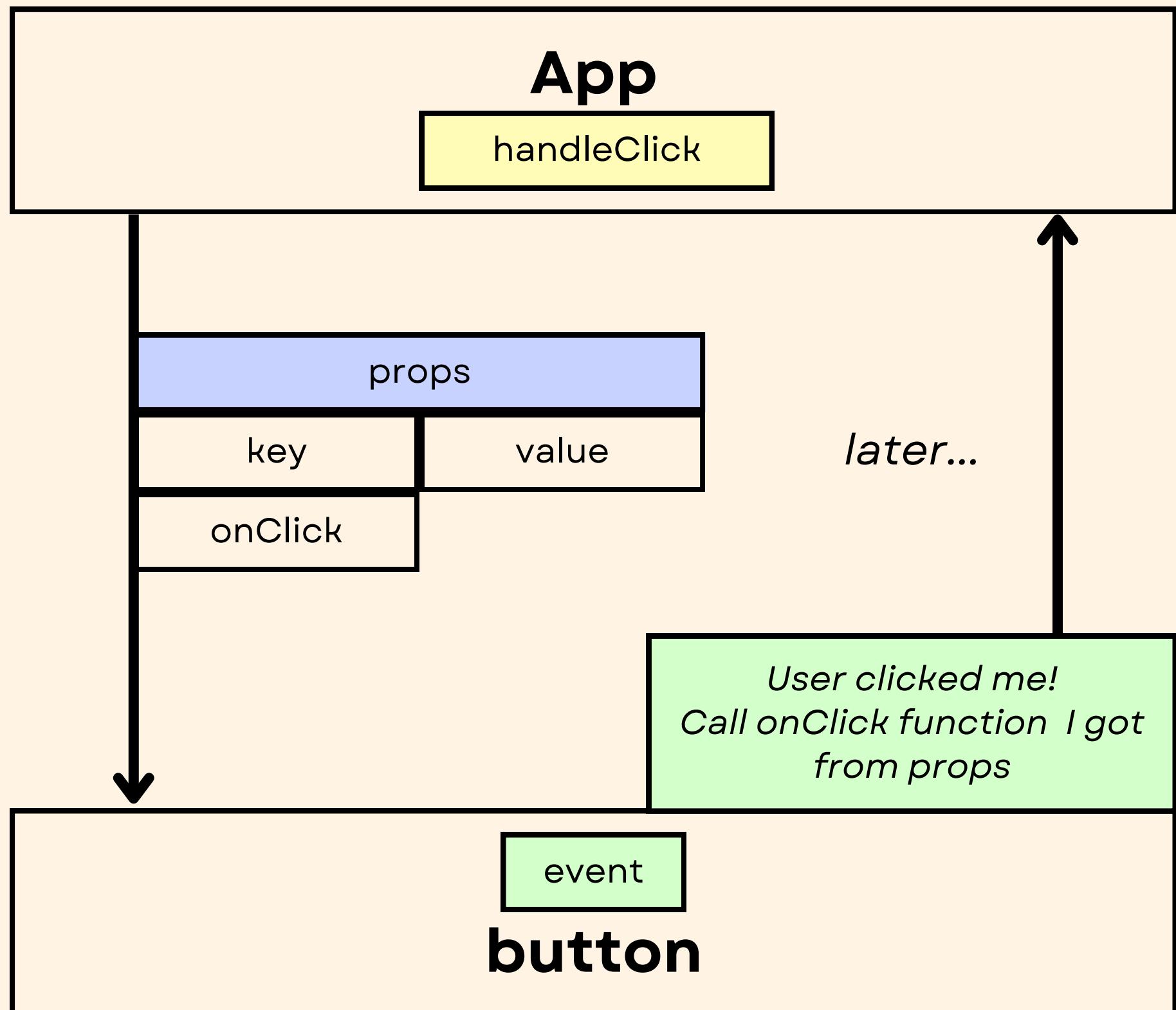


Normal Event Handler

Pass event handler down  
*using the props system*

Handler gets called with an  
event object

**Child to Parent**  
communication, kinda  
using props?

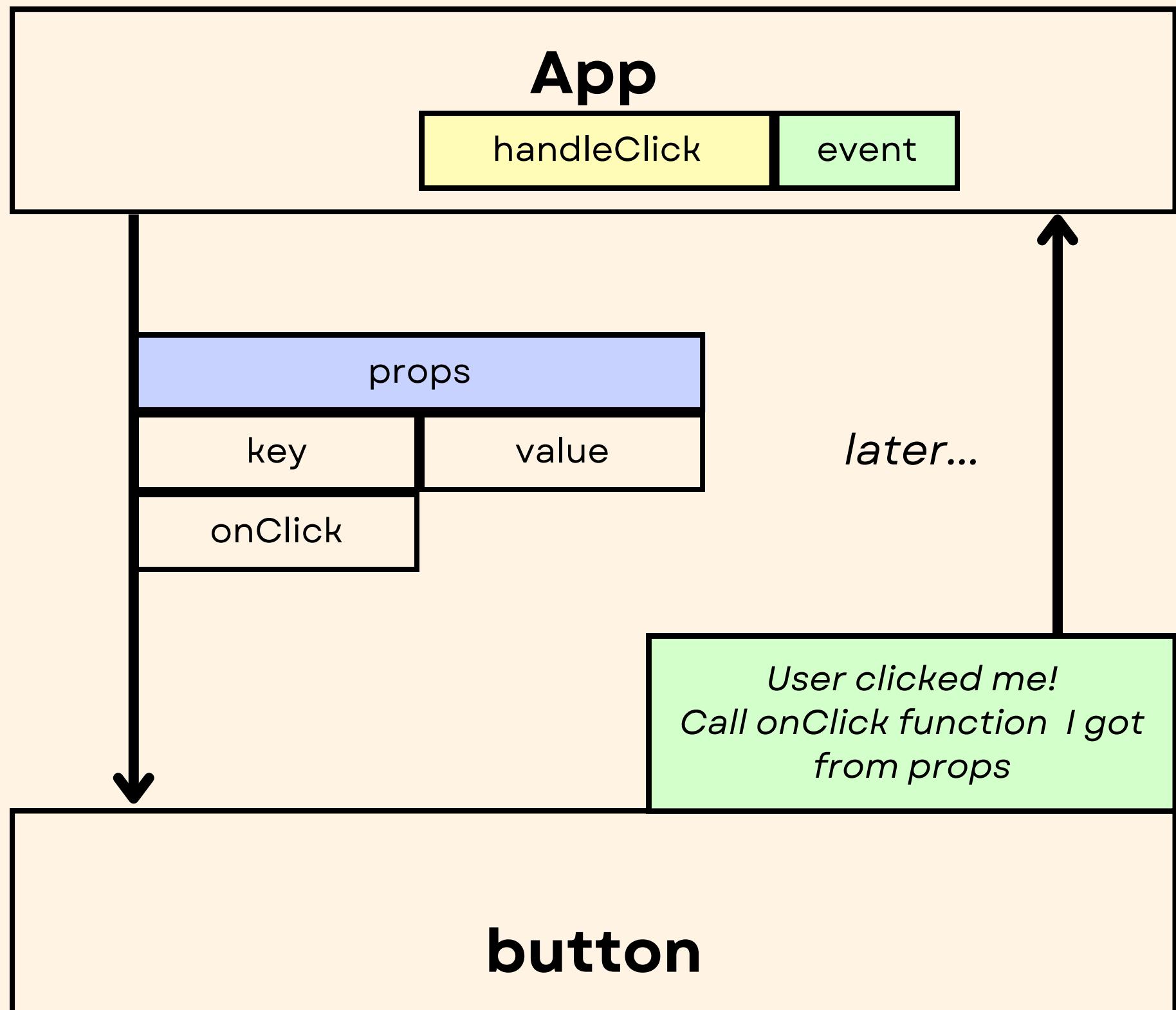


Normal Event Handler

Pass event handler down  
using the *props* system

Handler gets called with an  
event object

**Child to Parent**  
communication, kinda  
using props?



Normal Event Handler

Pass event handler down  
*using the props system*

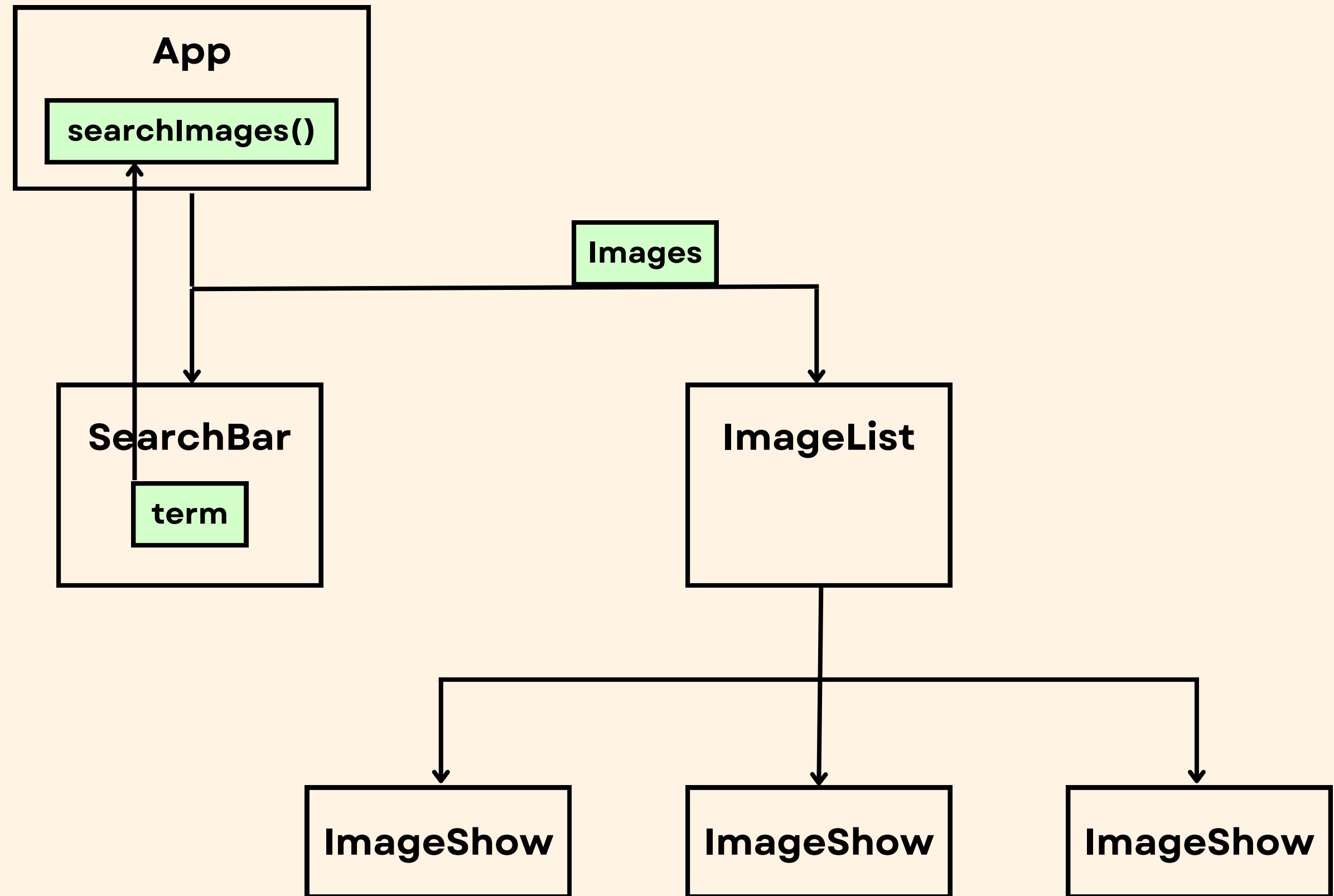
Handler gets called with an  
event object

**Child to Parent**  
communication, kinda  
using props?

**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

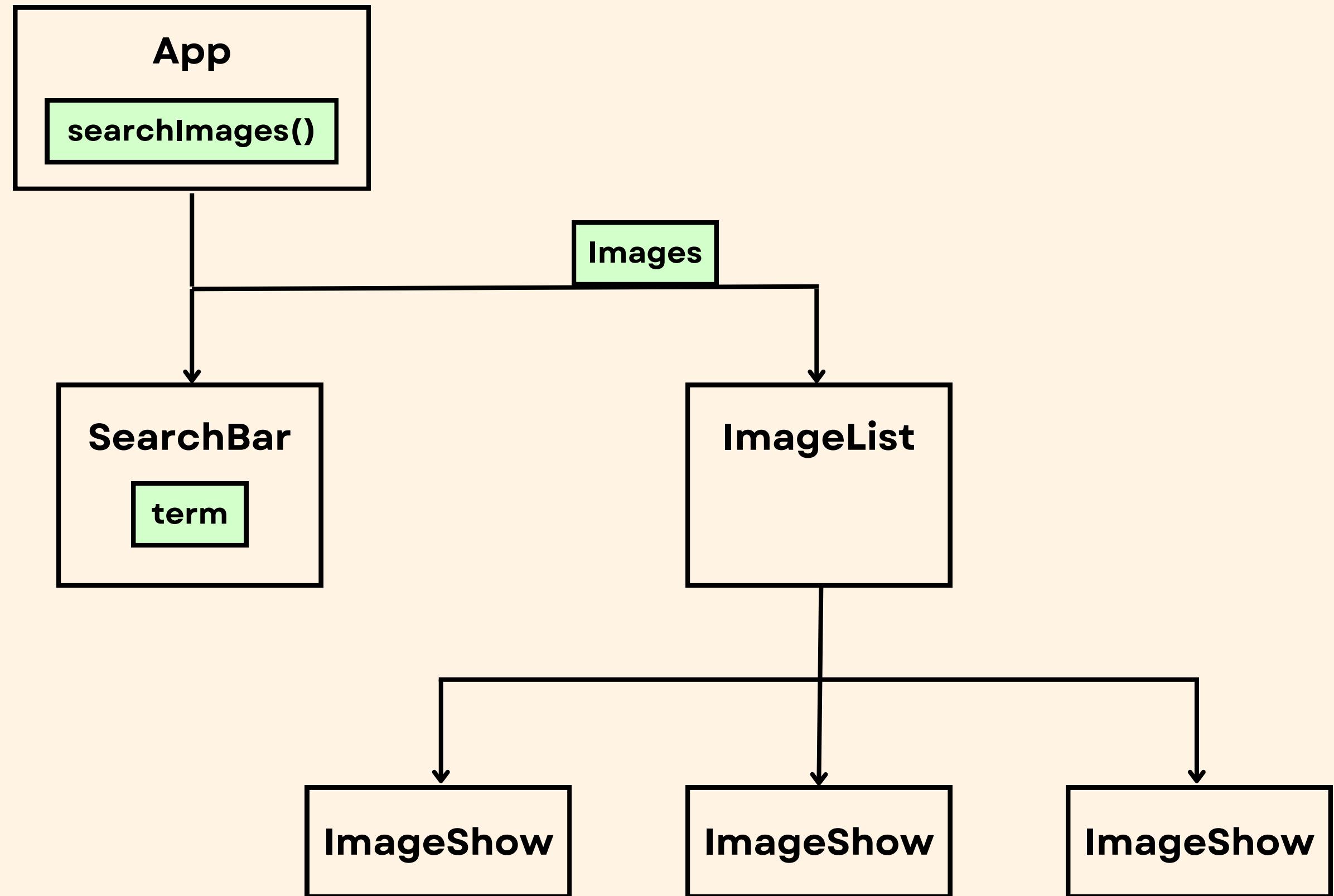
**????**



**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

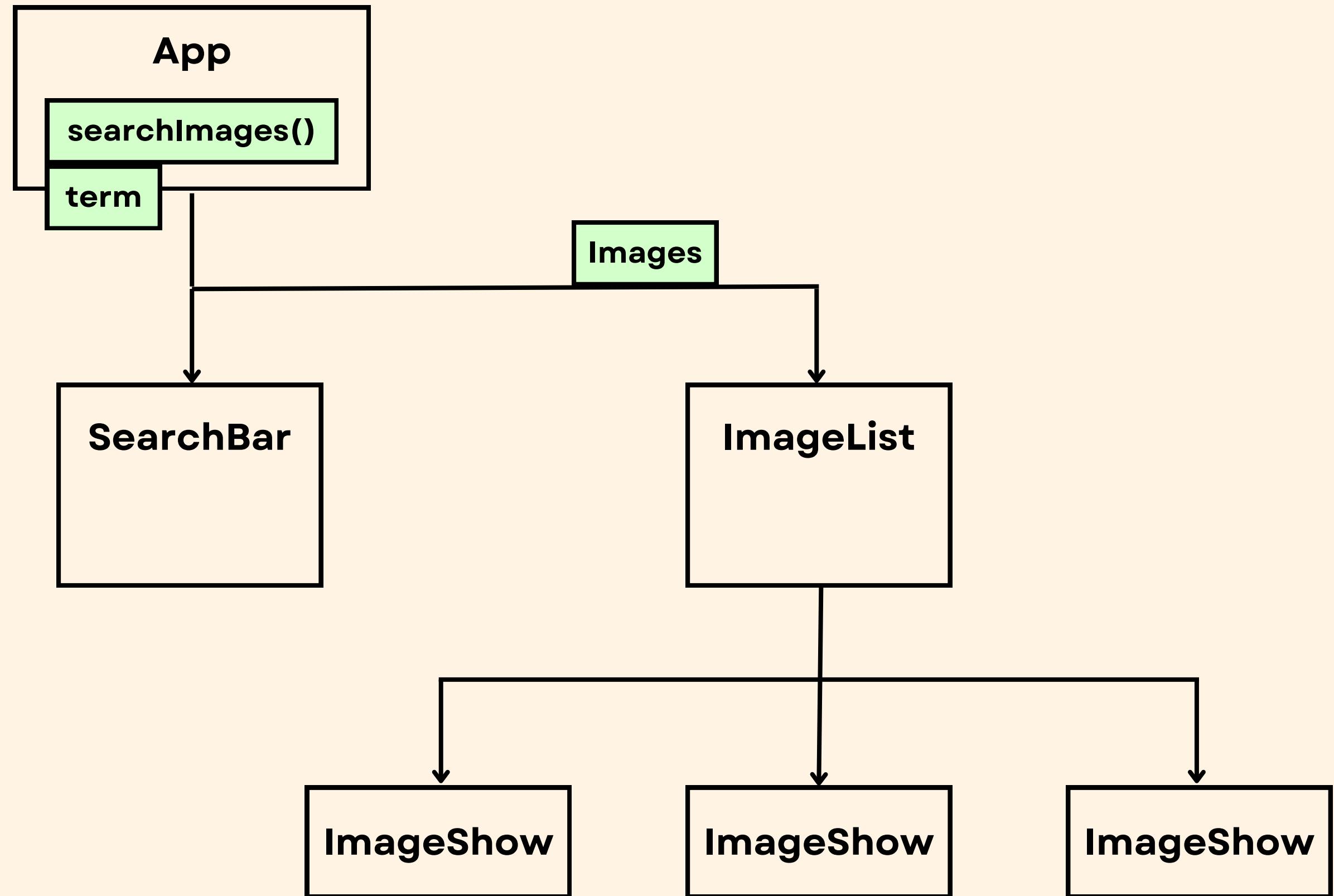
**????**



**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

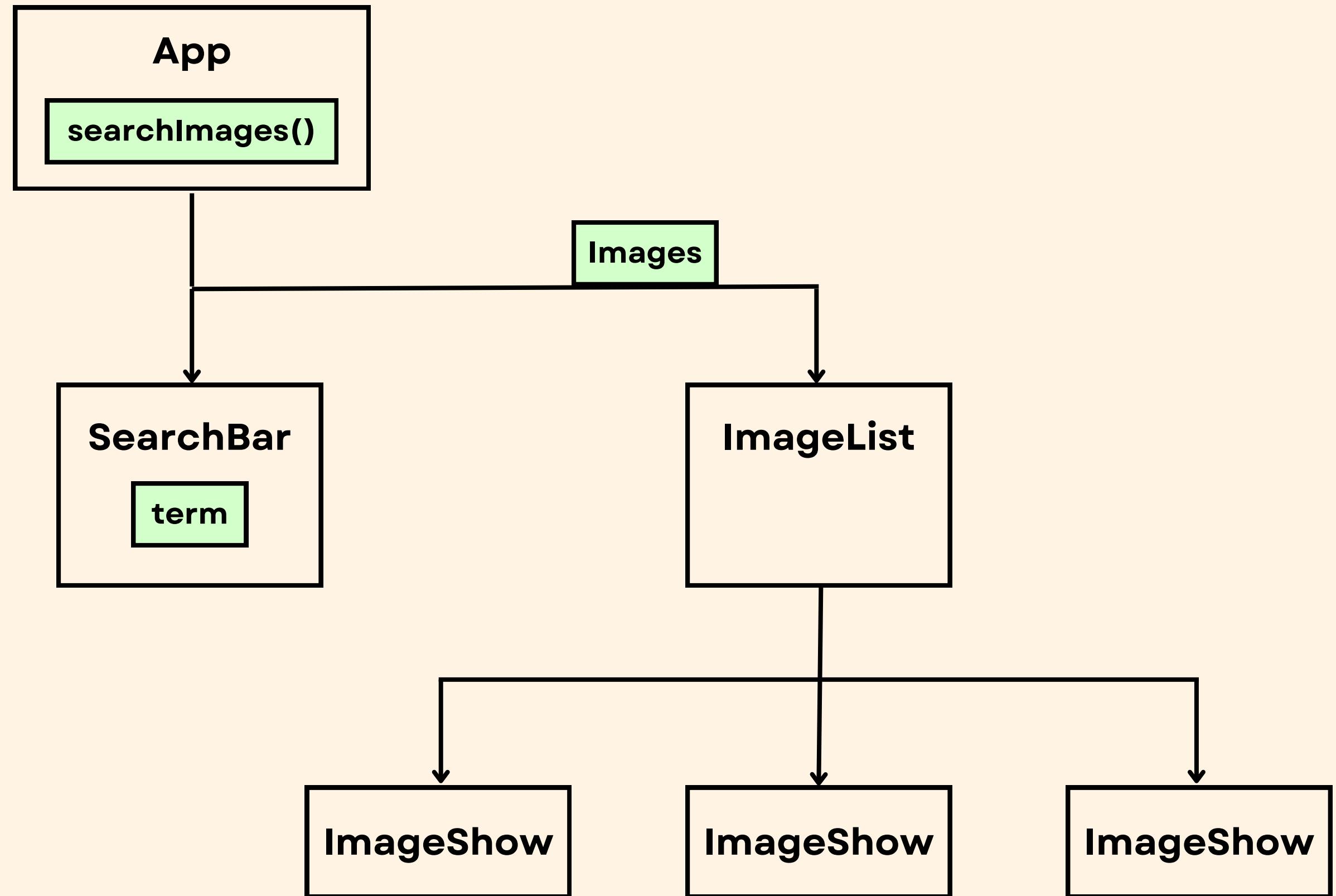
**????**

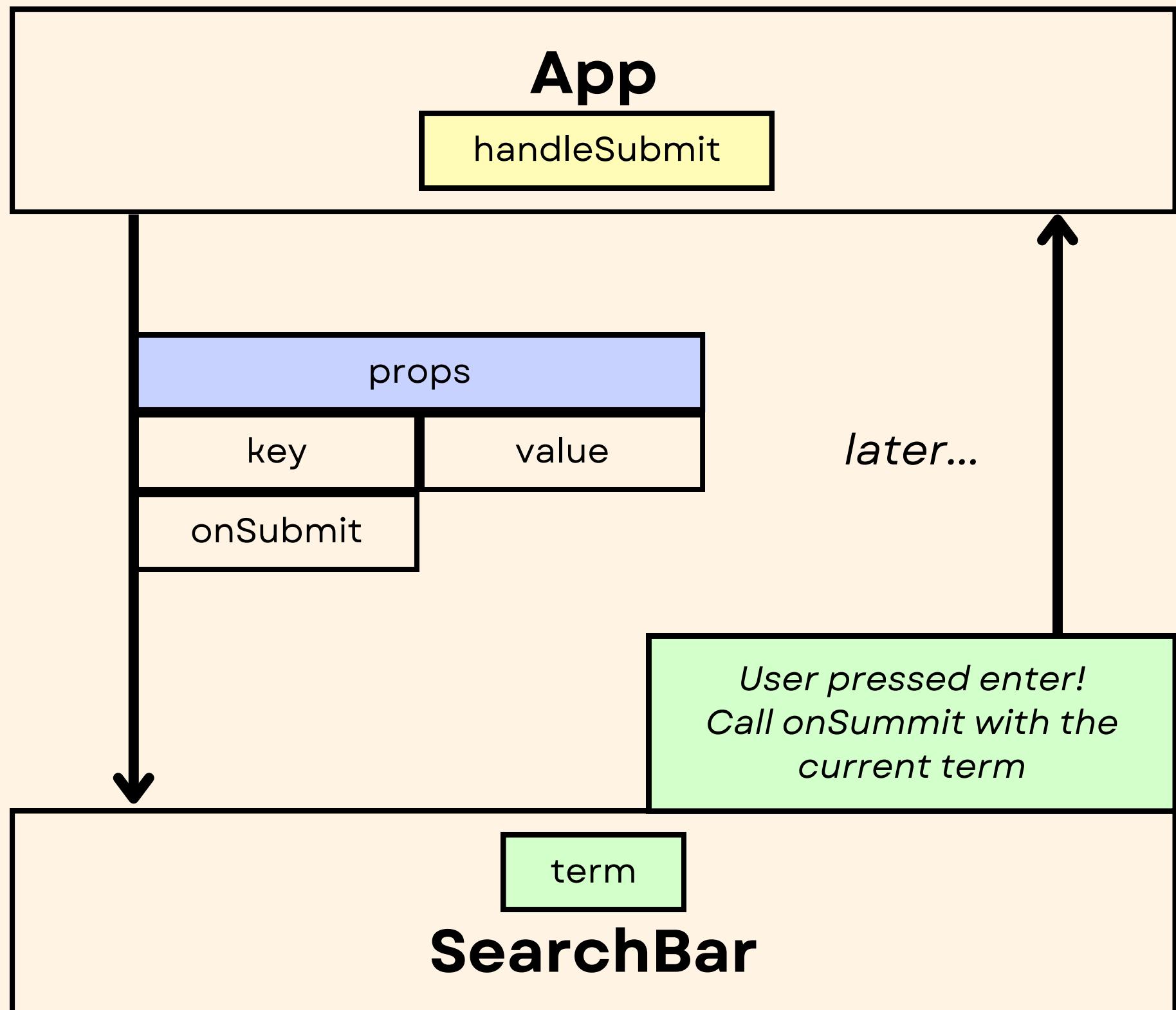


**When the user presses the  
'enter' key we need to get  
this search term up to the App**

**Child to parent  
communication**

**????**



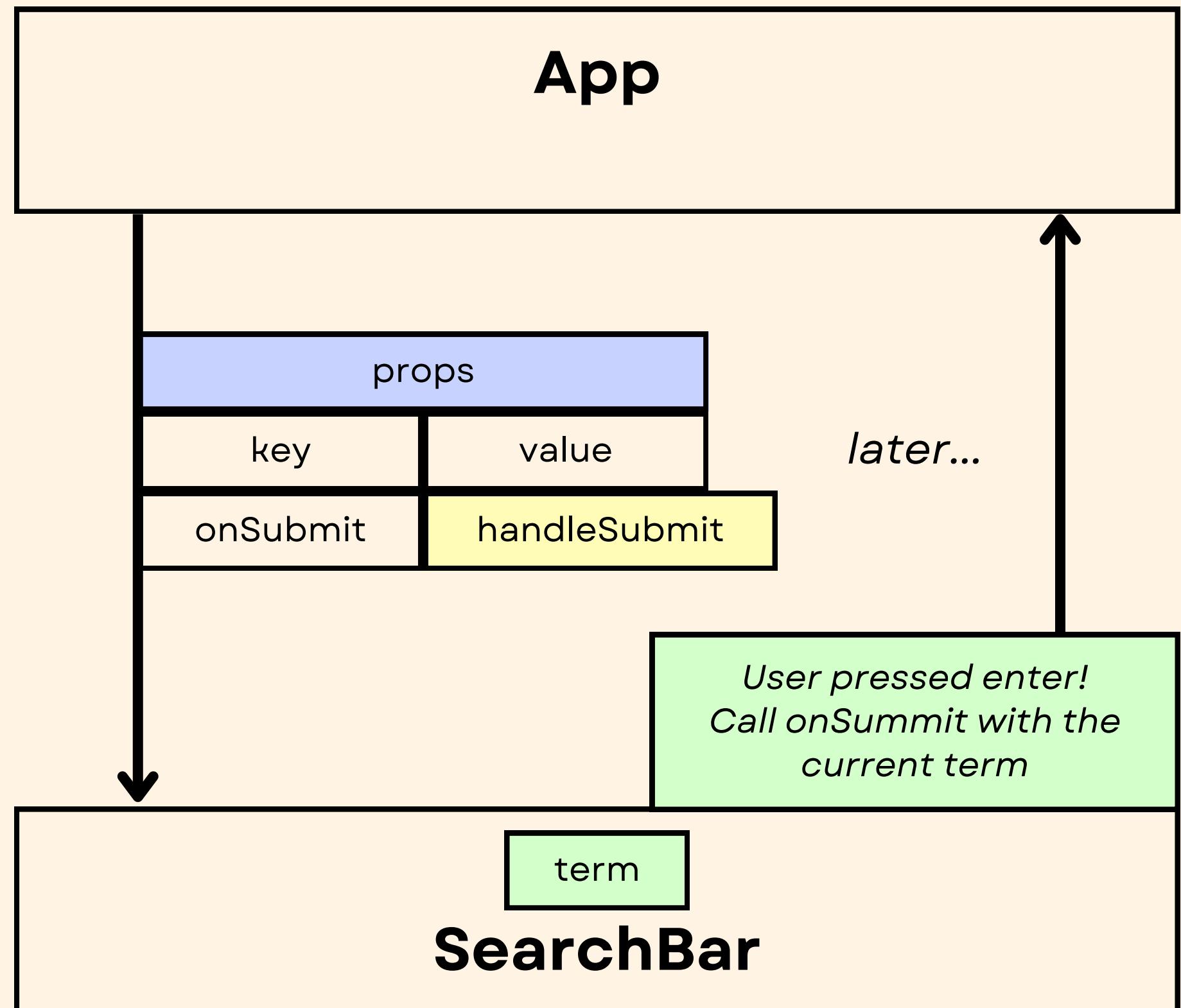


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

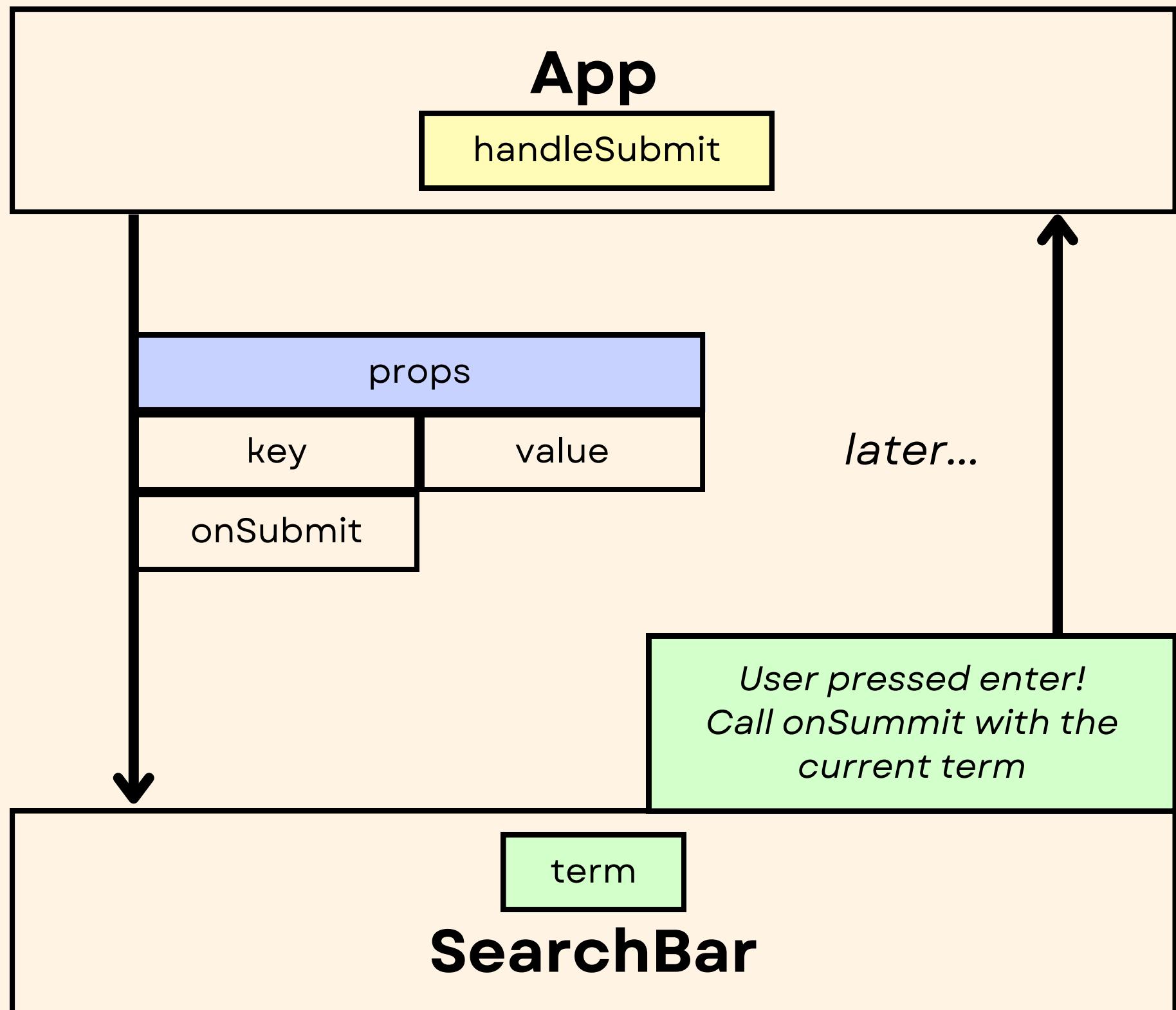


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

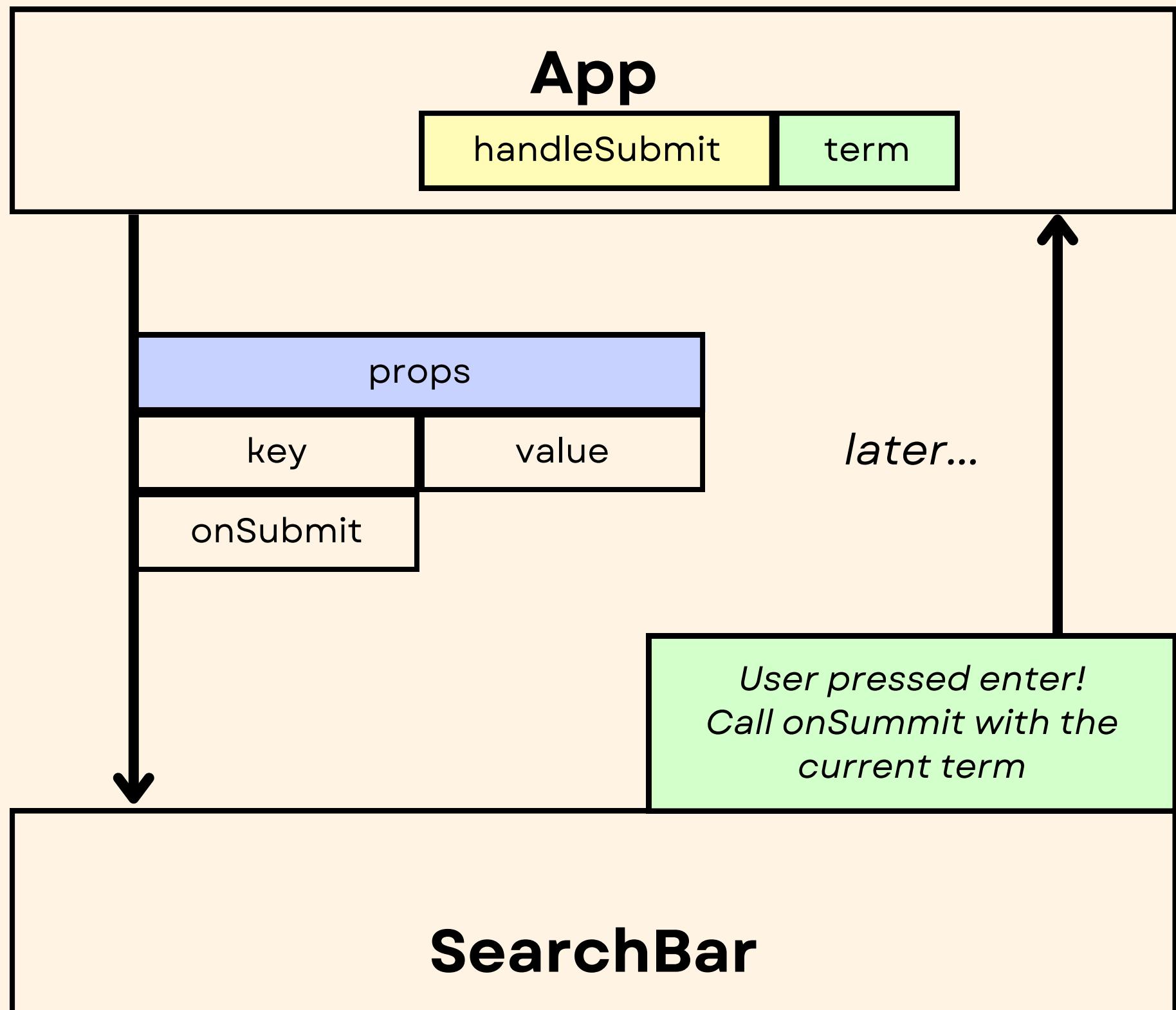


Communicate from Child **up** to Parent

**Treat it like a normal event!**

Pass an event handler down.

Call handler when something interesting happens

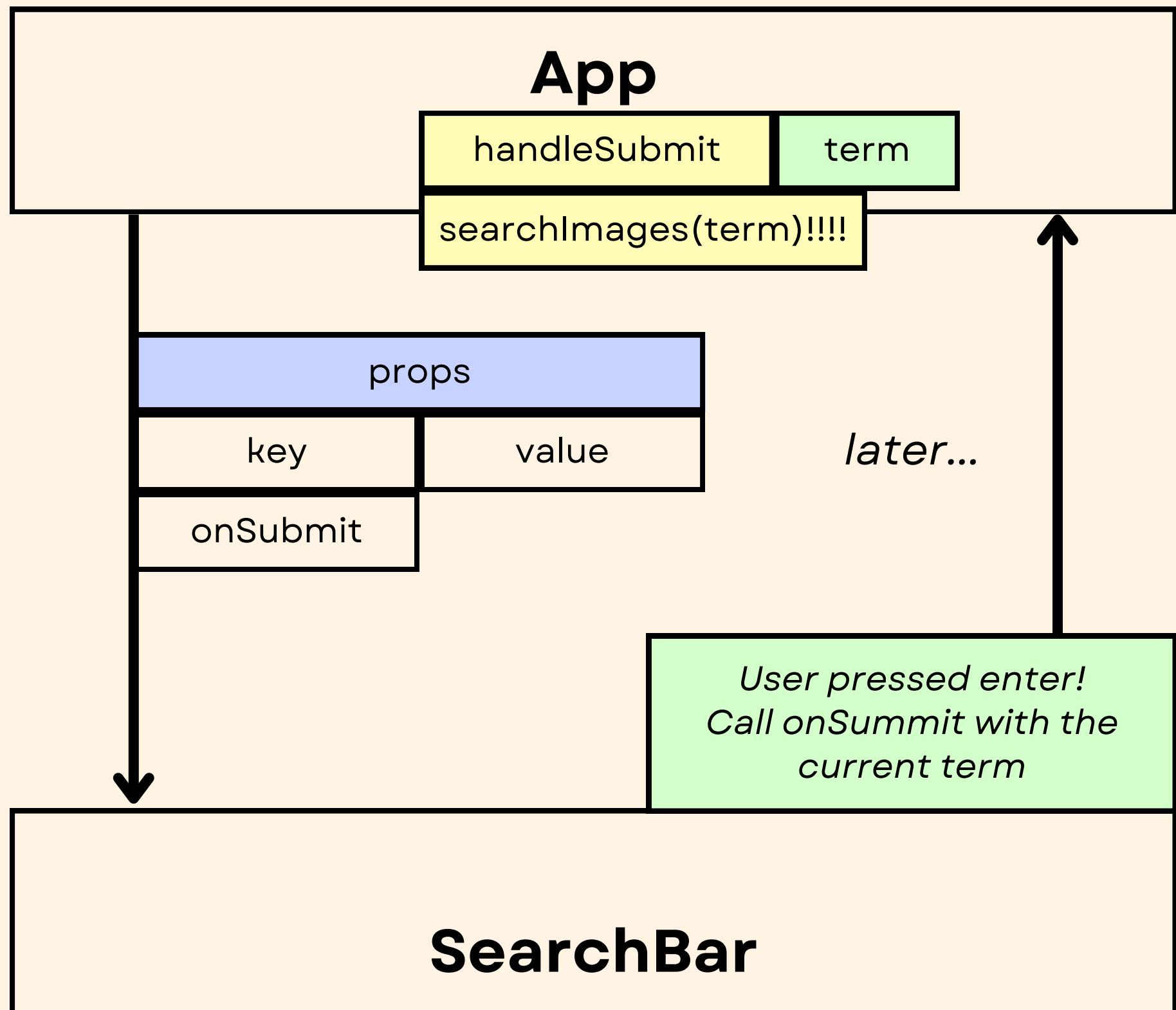


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

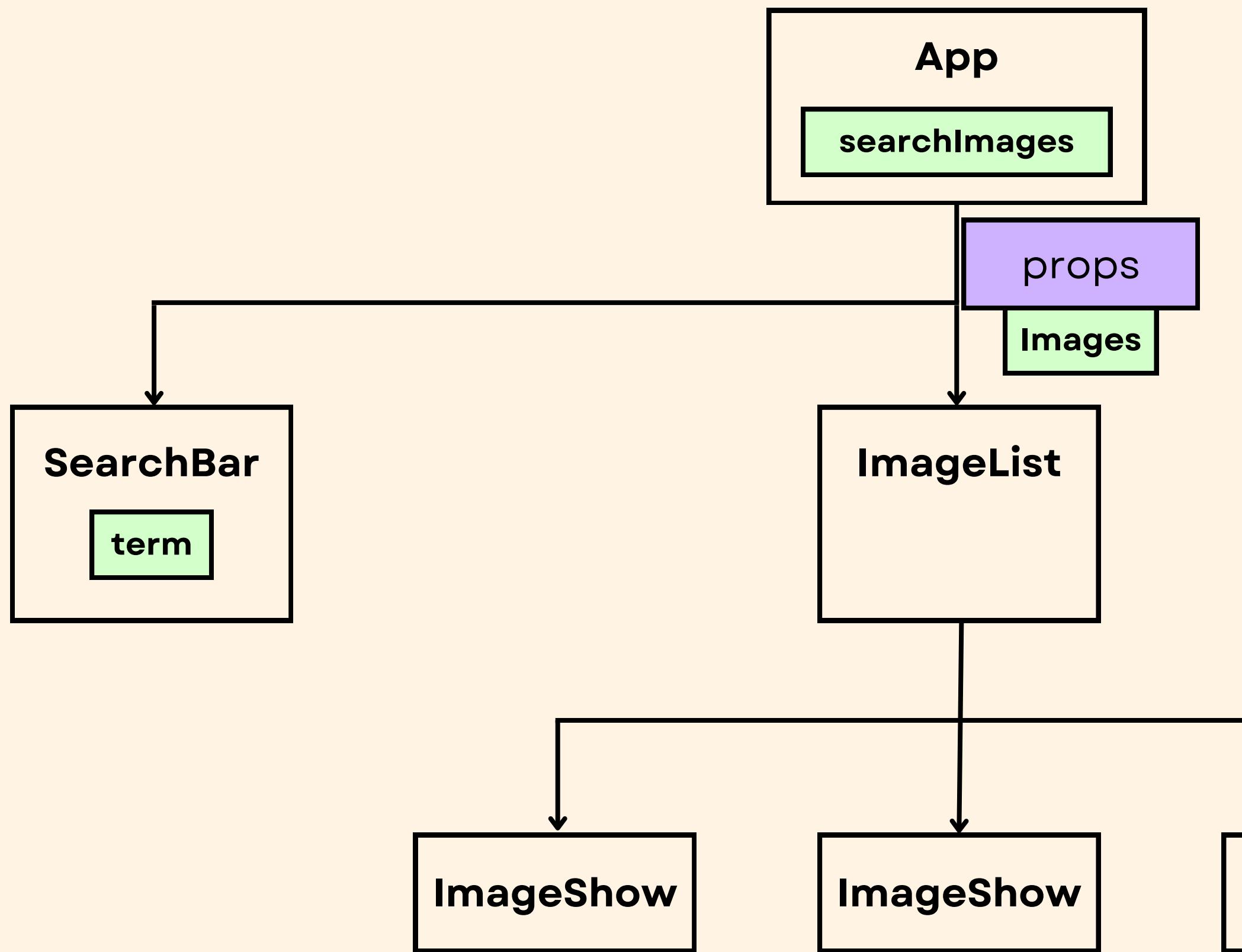


Communicate from Child **up** to Parent

**Treat it like a normal event!**

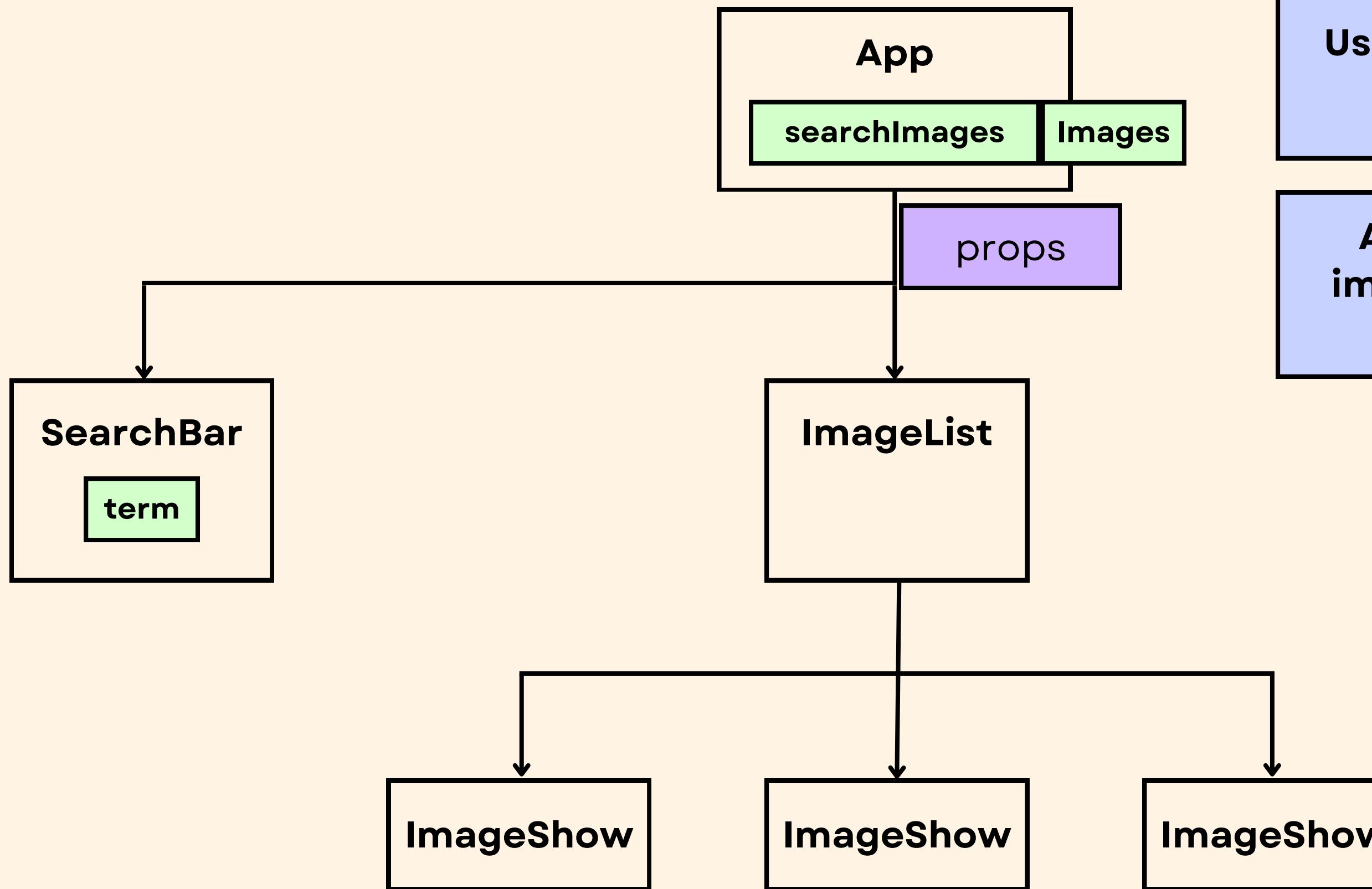
Pass an event handler down.

Call handler when something interesting happens



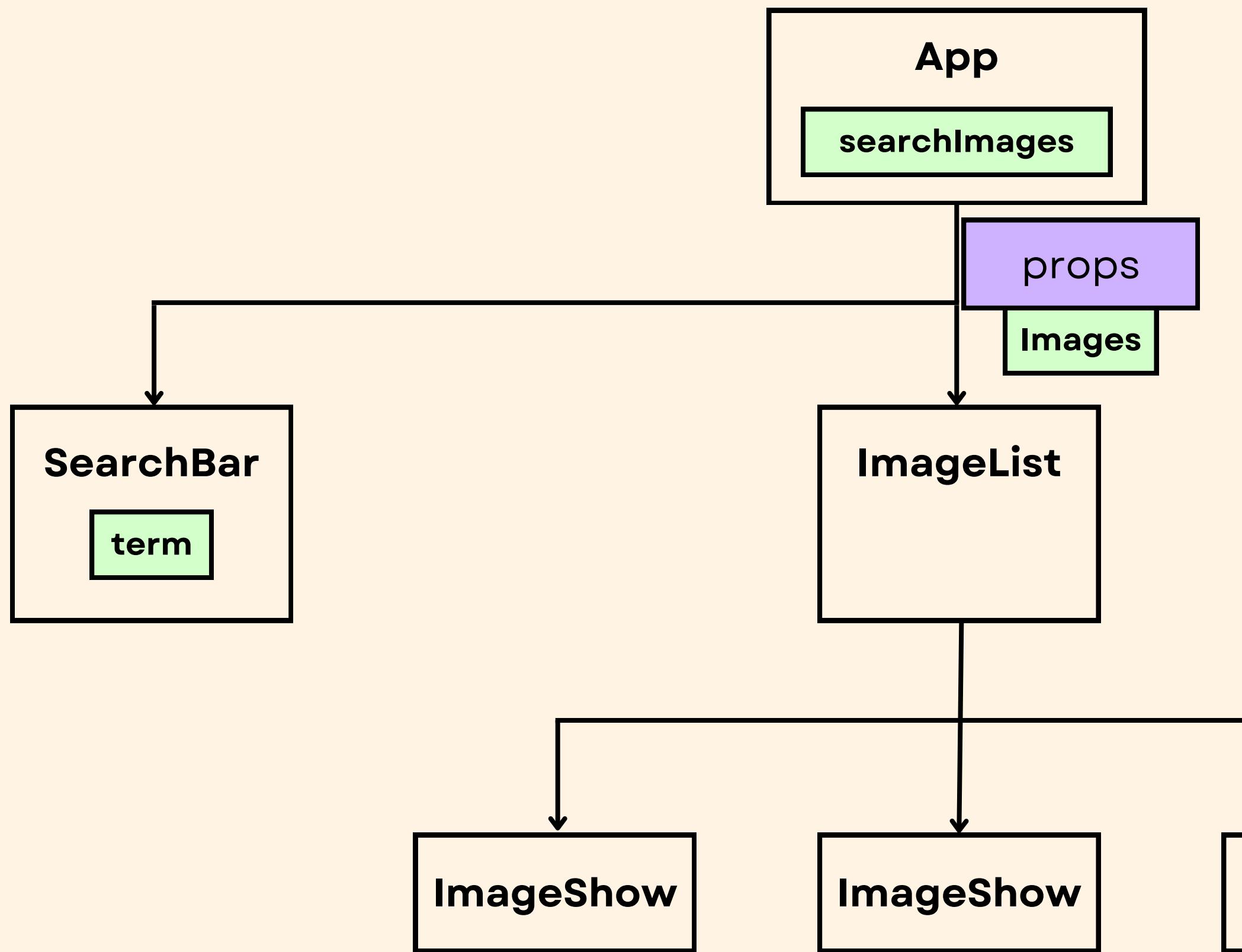
**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**



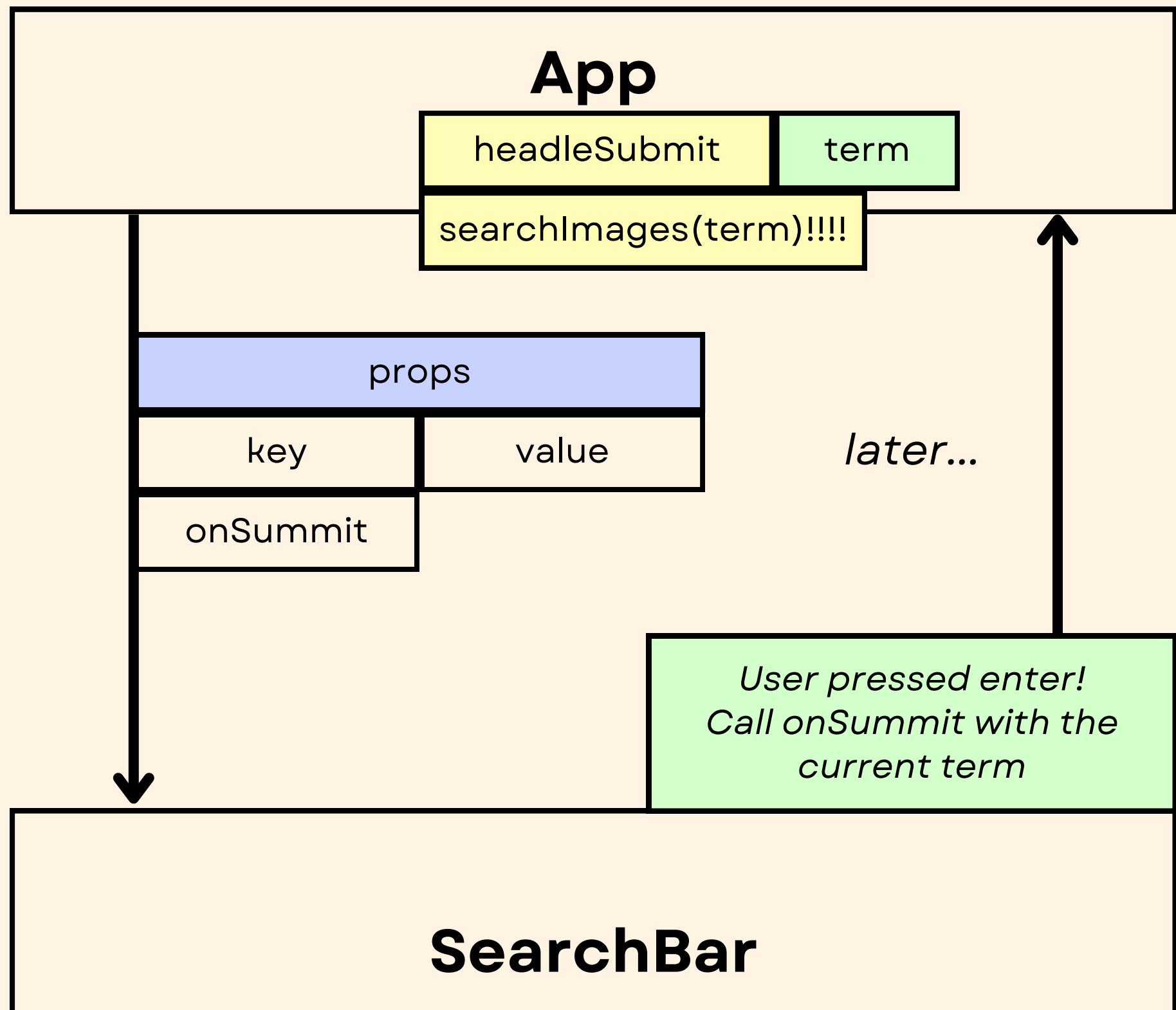
**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**



**Use props to communicate from parent to child**

**App can send the list of images down to ImageList using props!**

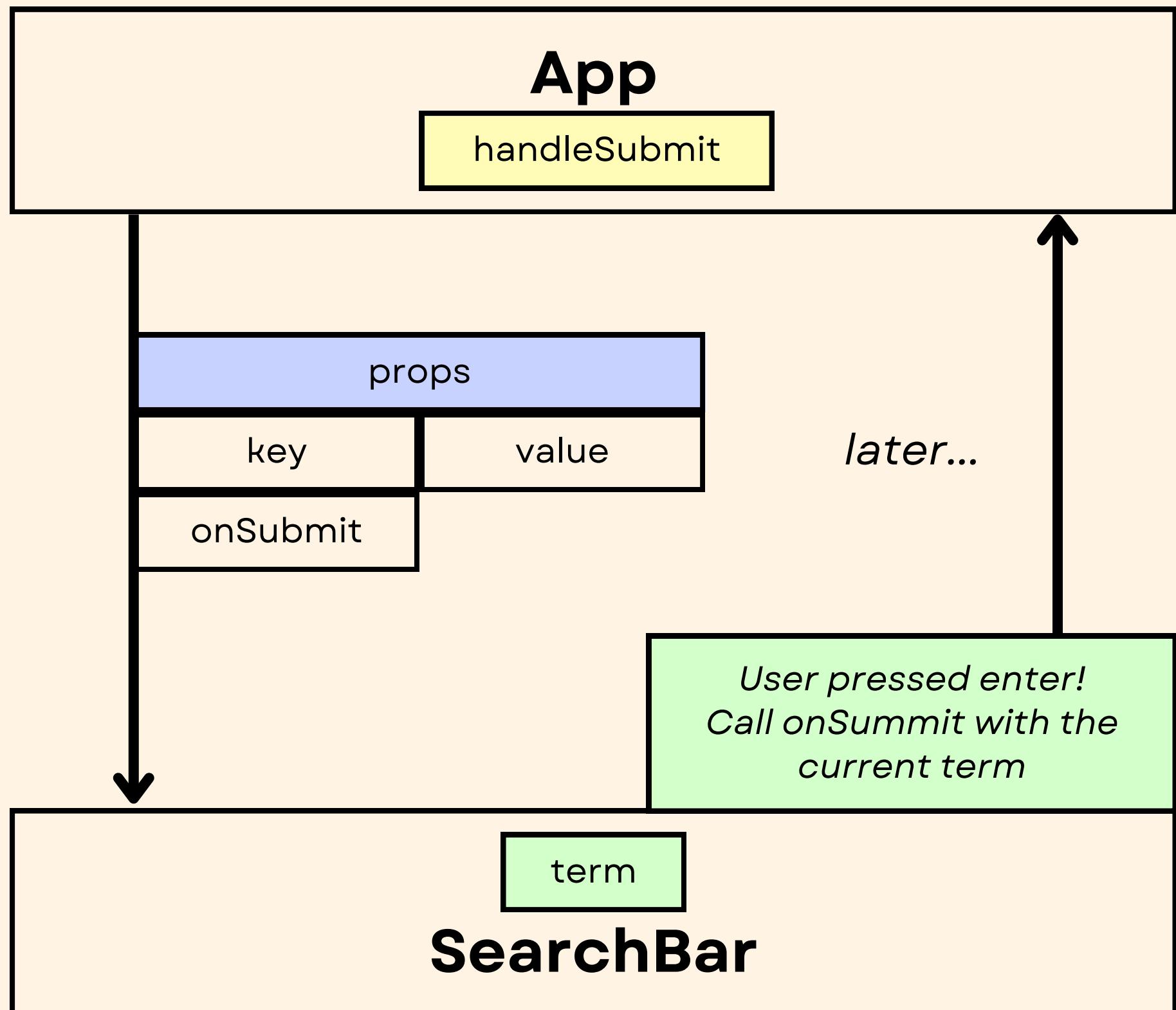


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

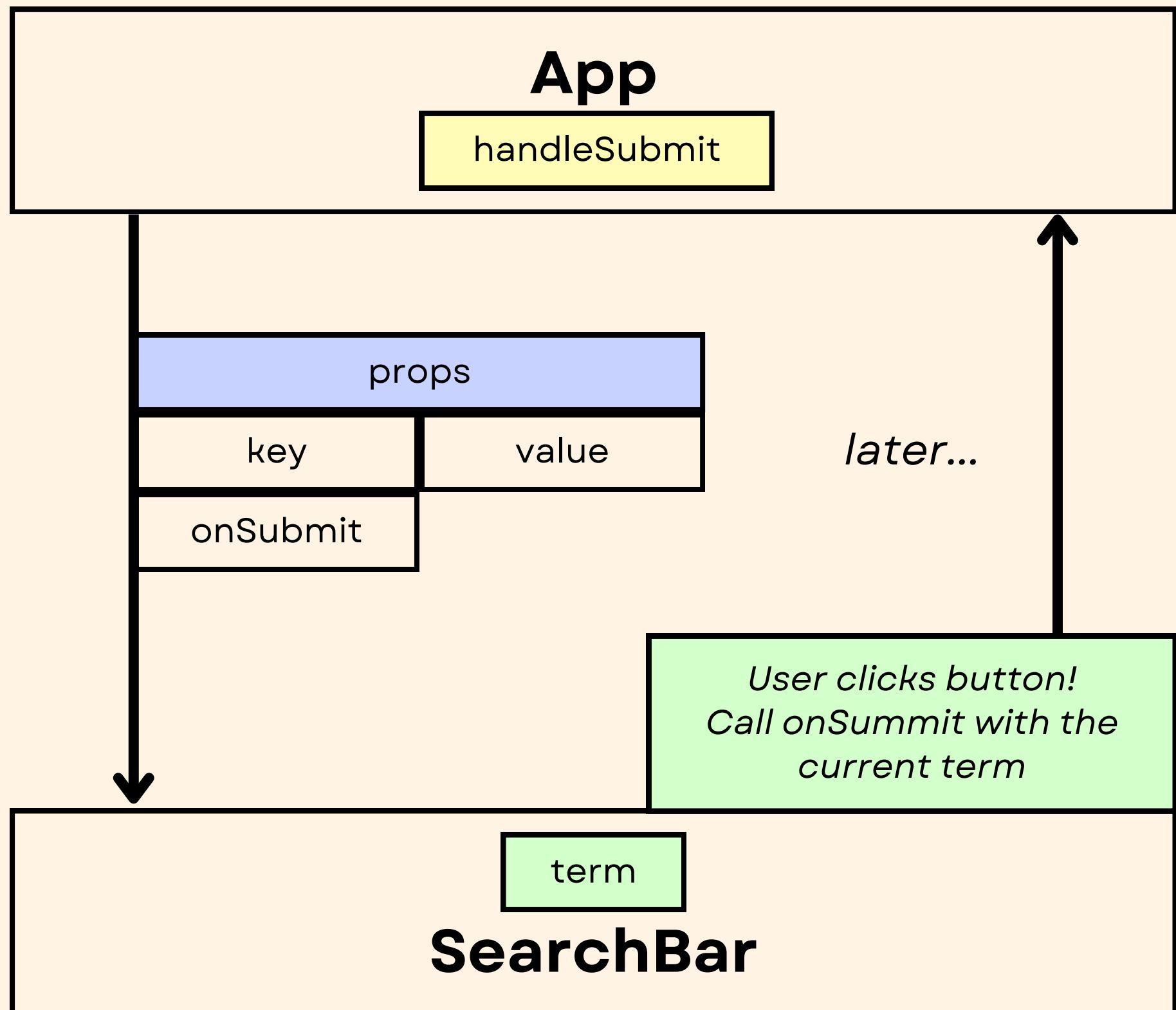


Communicate from Child **up** to Parent

**Treat it like a normal event!**

Pass an event handler down.

Call handler when something interesting happens

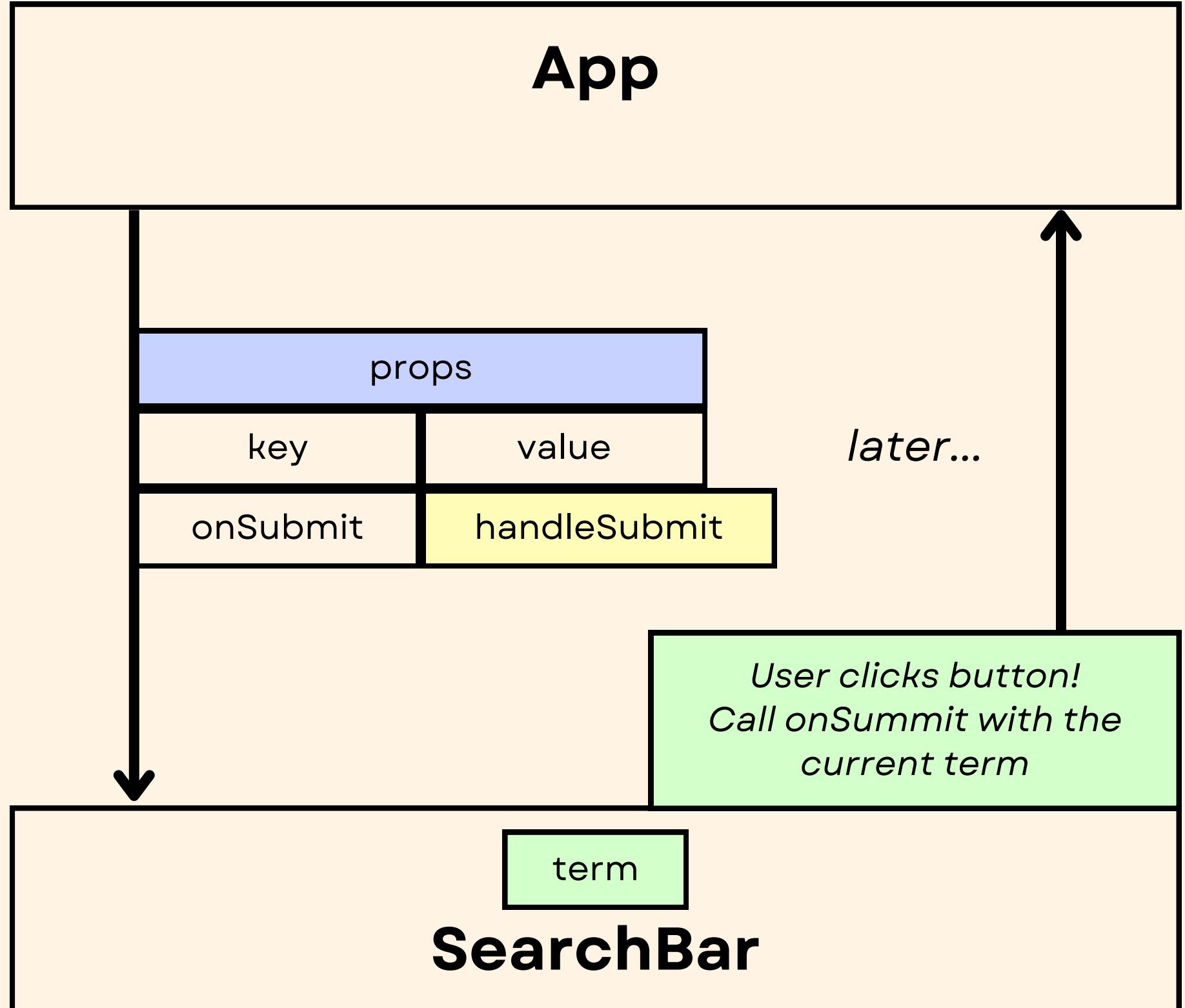


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

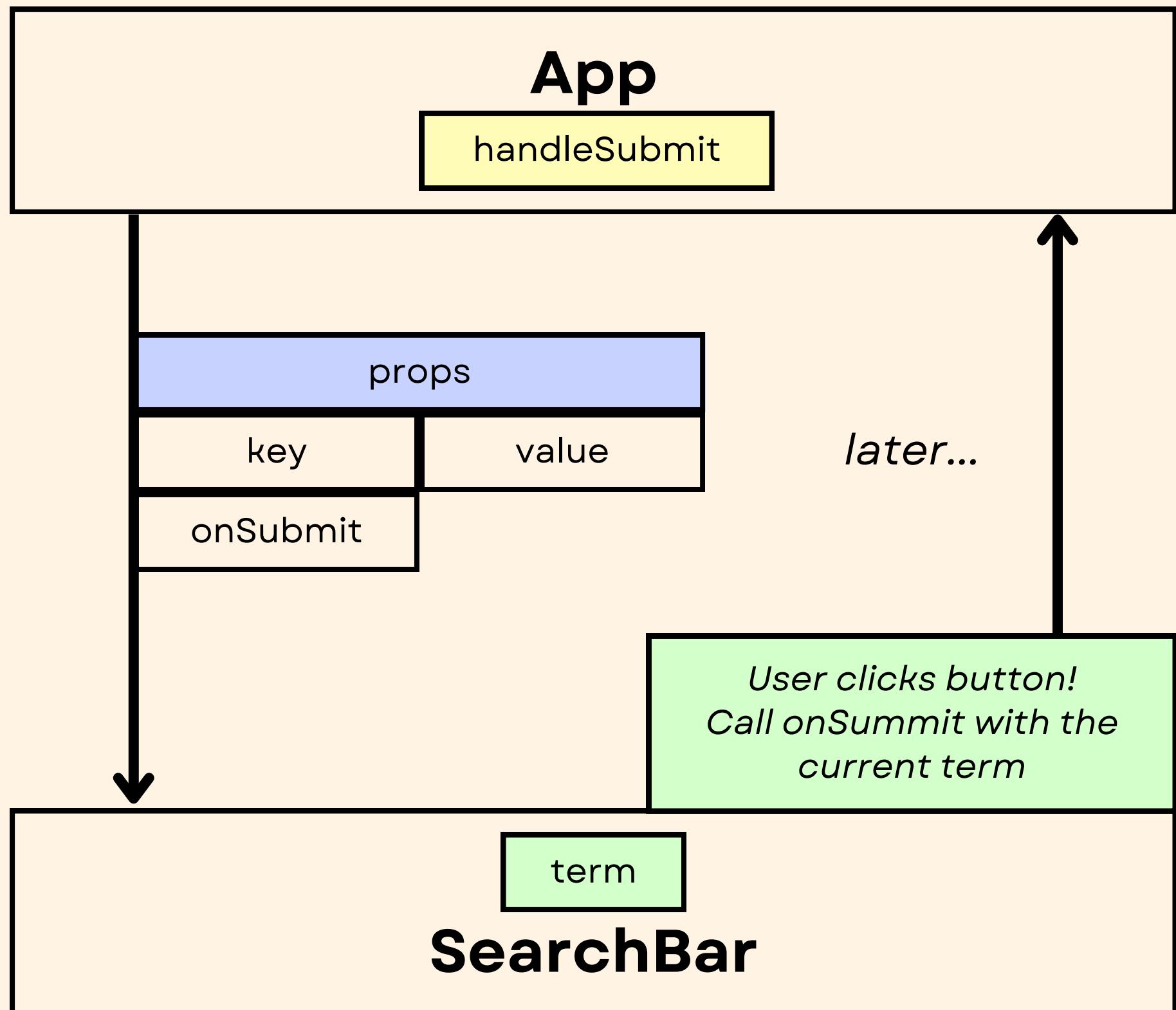


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

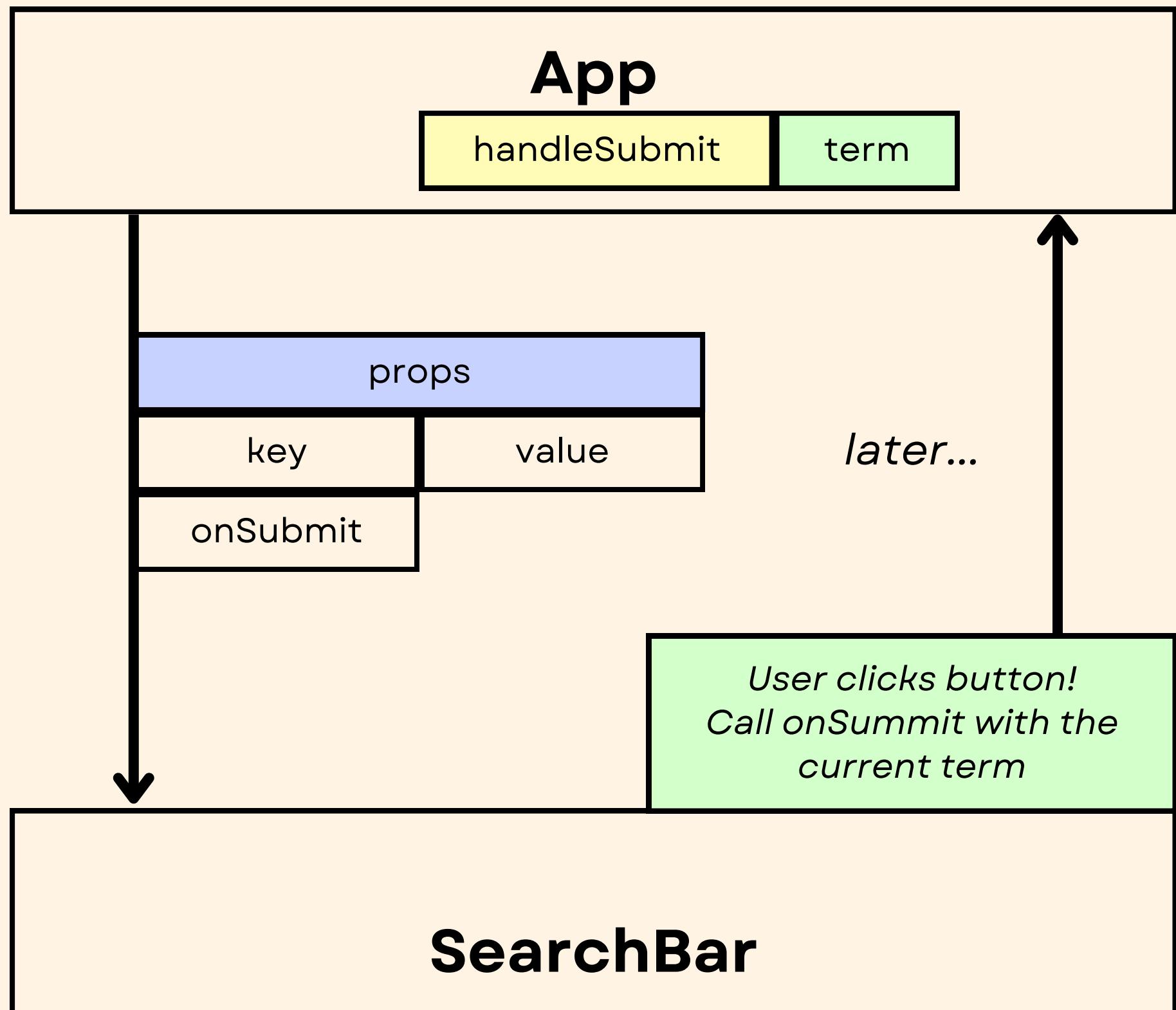


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

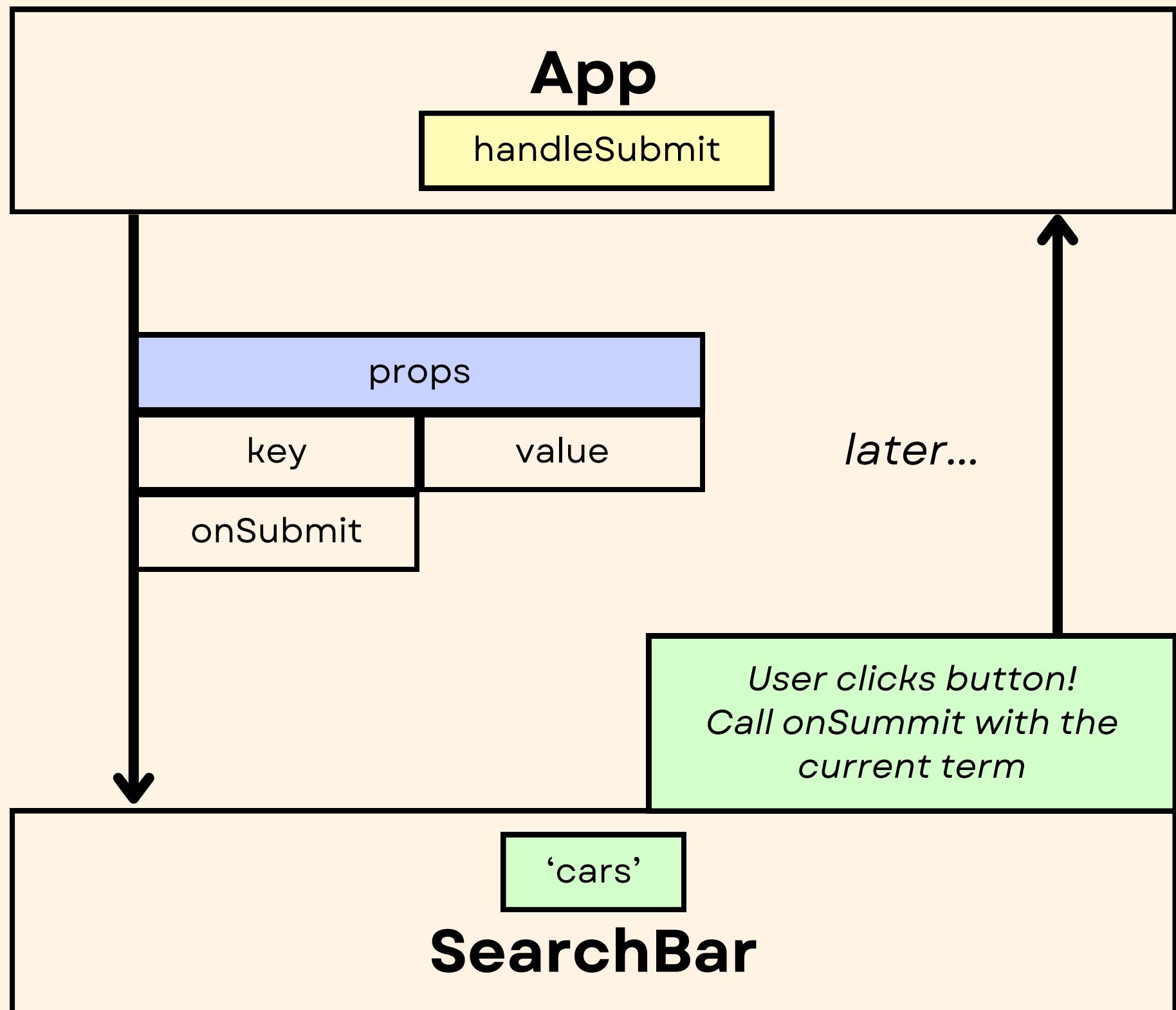


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens

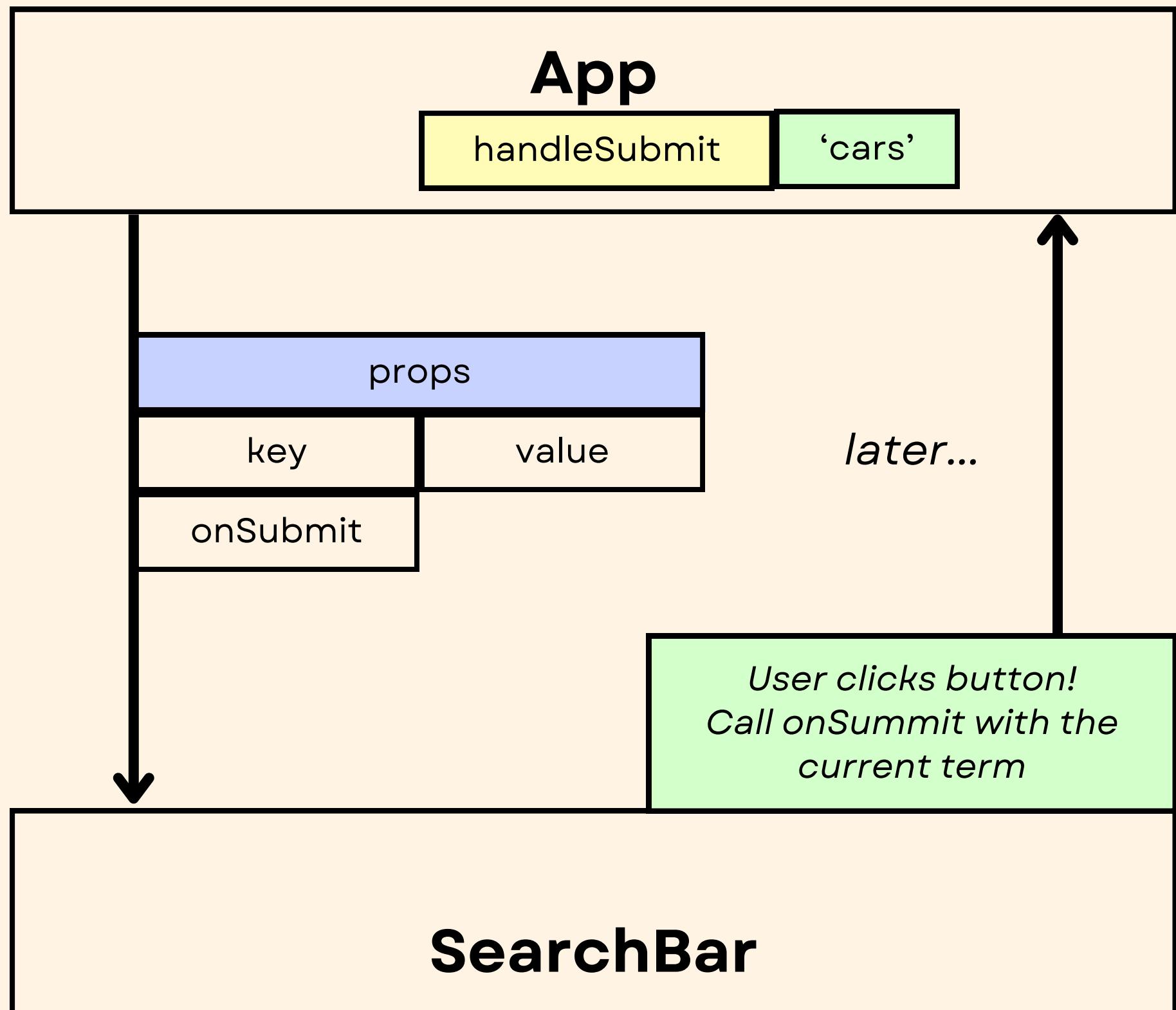


Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

Pass an event handler  
down.

Call handler when  
something interesting  
happens



Communicate from  
Child **up** to Parent

**Treat it like a normal  
event!**

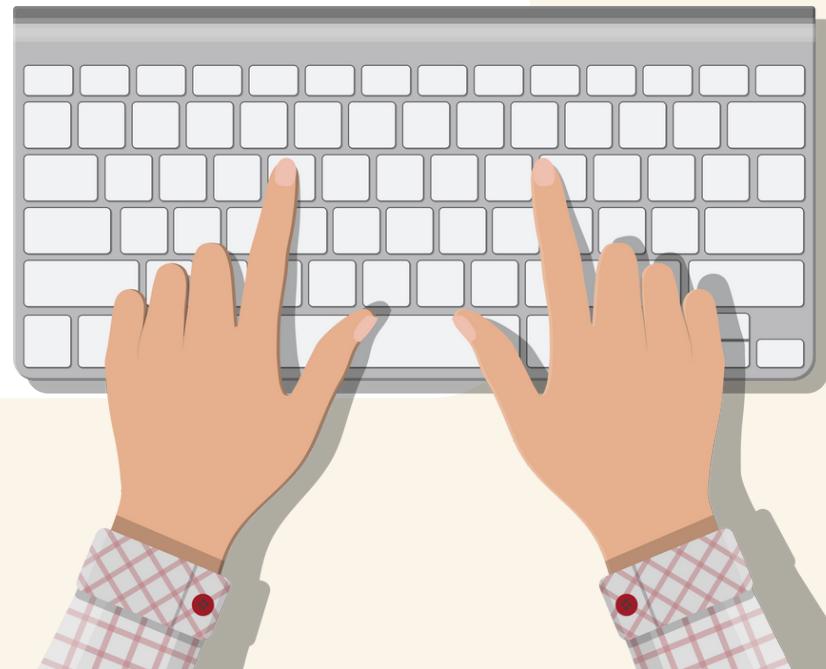
Pass an event handler  
down.

Call handler when  
something interesting  
happens

อีเมล

รหัส

ส่ง



```
<from>
  <label>Email</label>
  <input name="email" />

  <label>Password</label>
  <input name="password" />

  <button>Submit</button>
</from>
```

อีเมล

รหัส

ส่ง

myapp.com?email=asdf@asdf.com&password=asdf



```
<from>
  <label>Email</label>
  <input name="email" />

  <label>Password</label>
  <input name="password" />

  <button>Submit</button>
</from>
```

```
function SearchBar({ onSubmit }) {
  const handleFormSubmit = (event) => {
    event.preventDefault();

    onSubmit(document.querySelector('input').value);
  };

  return (
    <div>
      <form onSubmit={handleFormSubmit}>
        <input />
      </form>
    </div>
  );
}
```

อย่าทำแบบนี้  
เด็ดขาด

อย่าพยายามดึงค่าจาก input โดยใช้  
query selector (หรือวิธีที่คล้ายกัน  
ที่ใช้ Document Object)

คุณจะไม่ผ่านการสัมภาษณ์งาน  
ถ้าคุณเขียนโค้ดแบบนี้



# คำเตือน!

วิธีที่ React จัดการองค์ประกอบฟอร์ม (Text field, Checkbox, Radio Button) มีความ  
เปลกประหลาดเล็กน้อย

# การจัดการข้อความ Inputs

1. สร้าง State ขึ้นมาใหม่

2. สร้าง event handler เพื่อสำหรับดัก Event  
'onChange'

3. ในการดึงค่าจาก input ใช้ Event ที่ได้จาก  
'onChange'

4. นำค่านั้นมาอัปเดตใน State

5. ส่ง State ไปยัง input ใน property ที่ชื่อว่า  
'value'

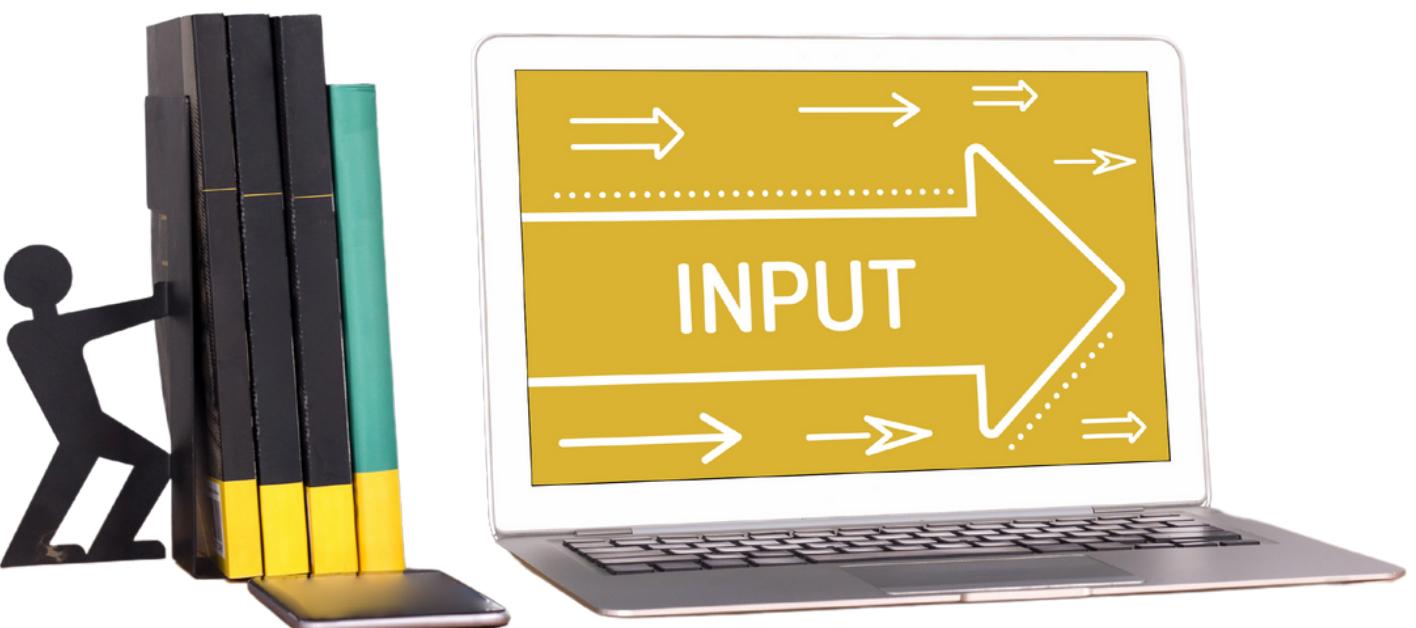
บังคับให้ input แสดงข้อความ "hi there!"  
อยู่เสมอและไม่ใช่ข้อความอื่น

`<input value="hi there!" />`

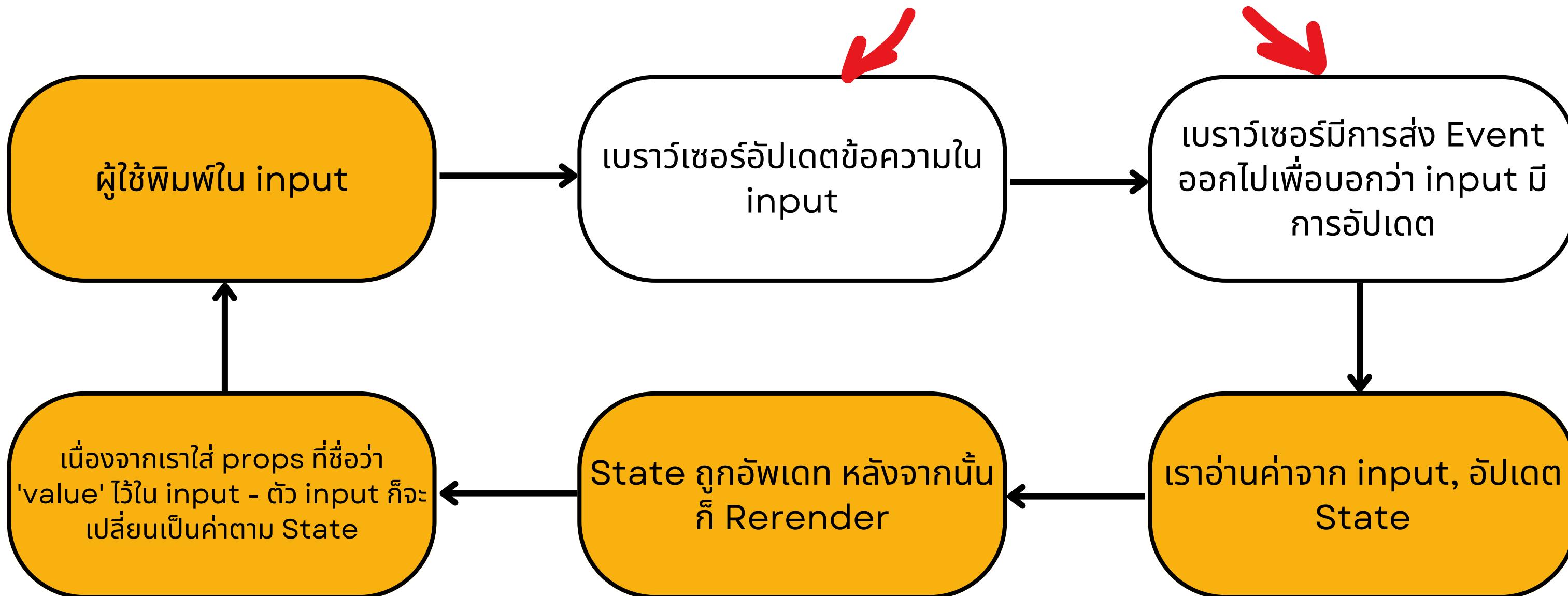


บังคับให้ input แสดงข้อความ "hi there!"  
อยู่เสมอและไม่ใช่ข้อความอื่น

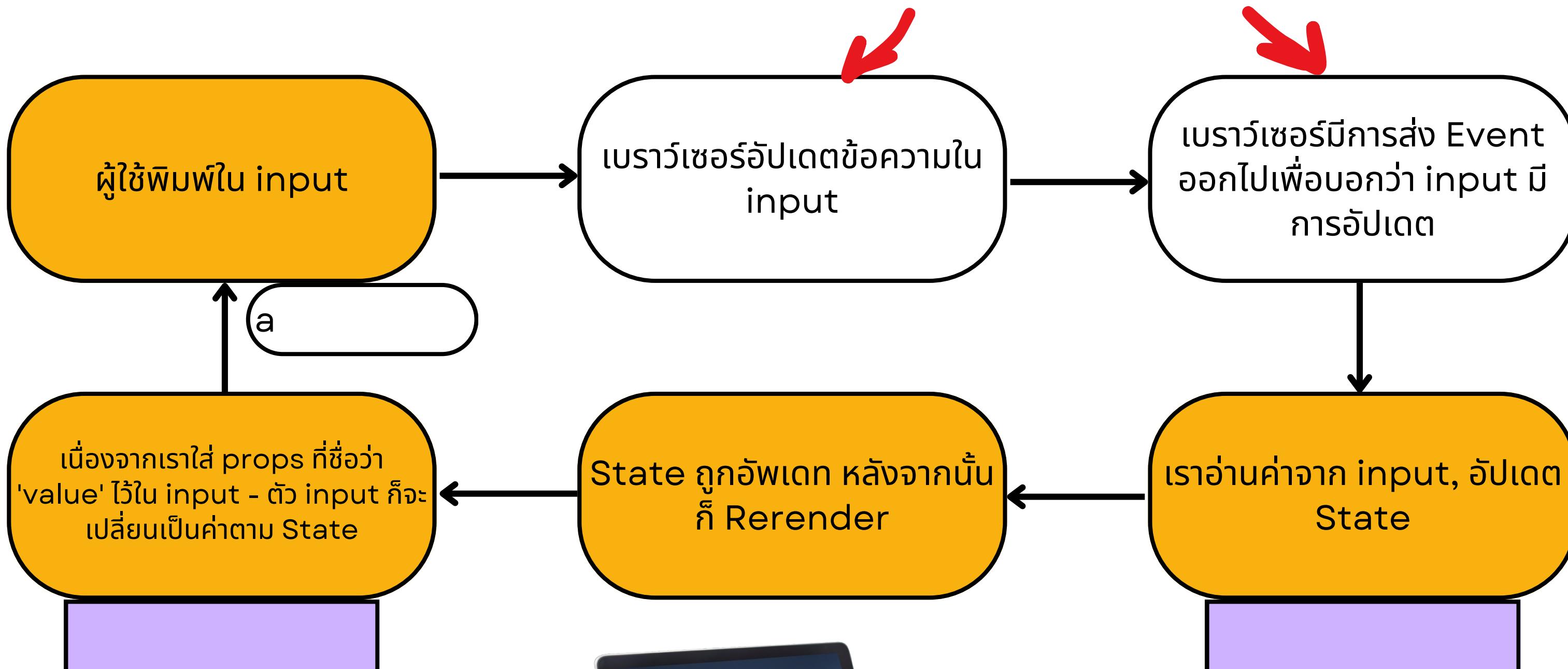
`<input value={term} />`



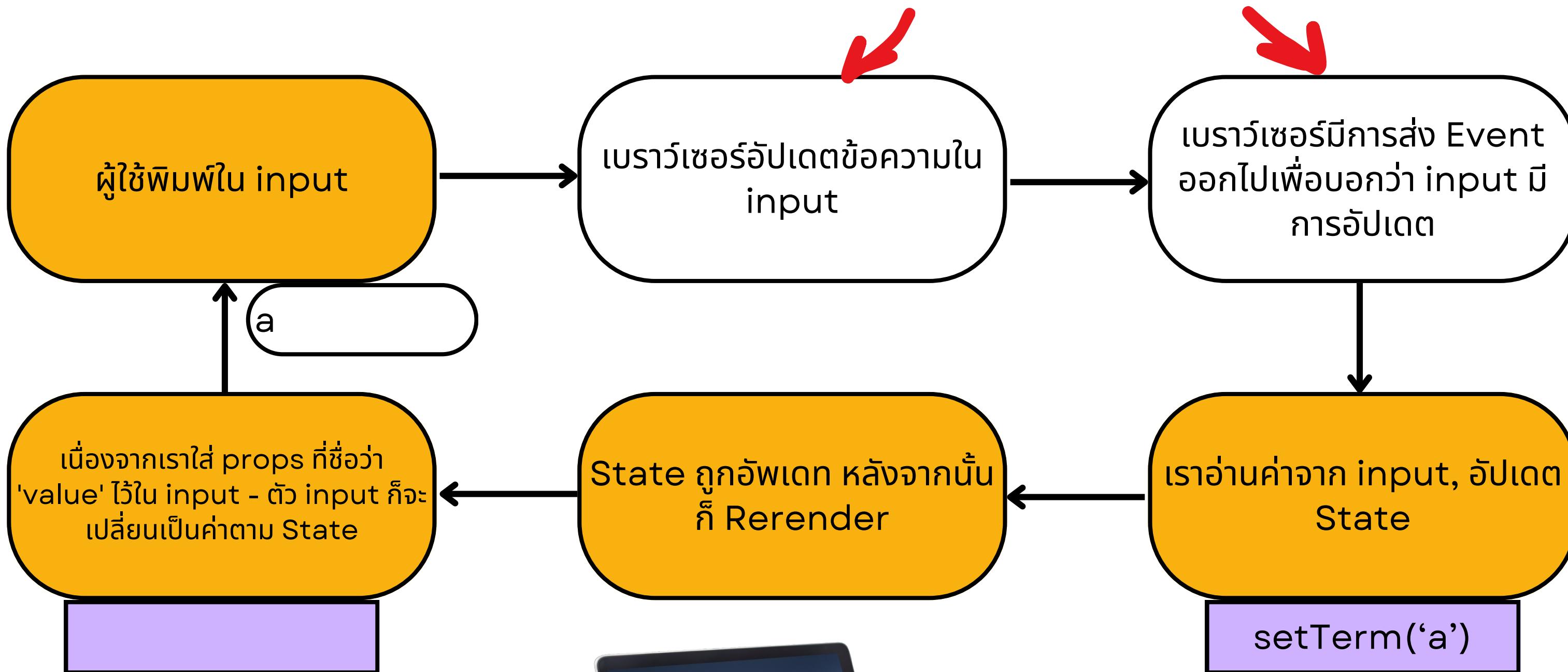
# เบราว์เซอร์เป็นคนจัดการ



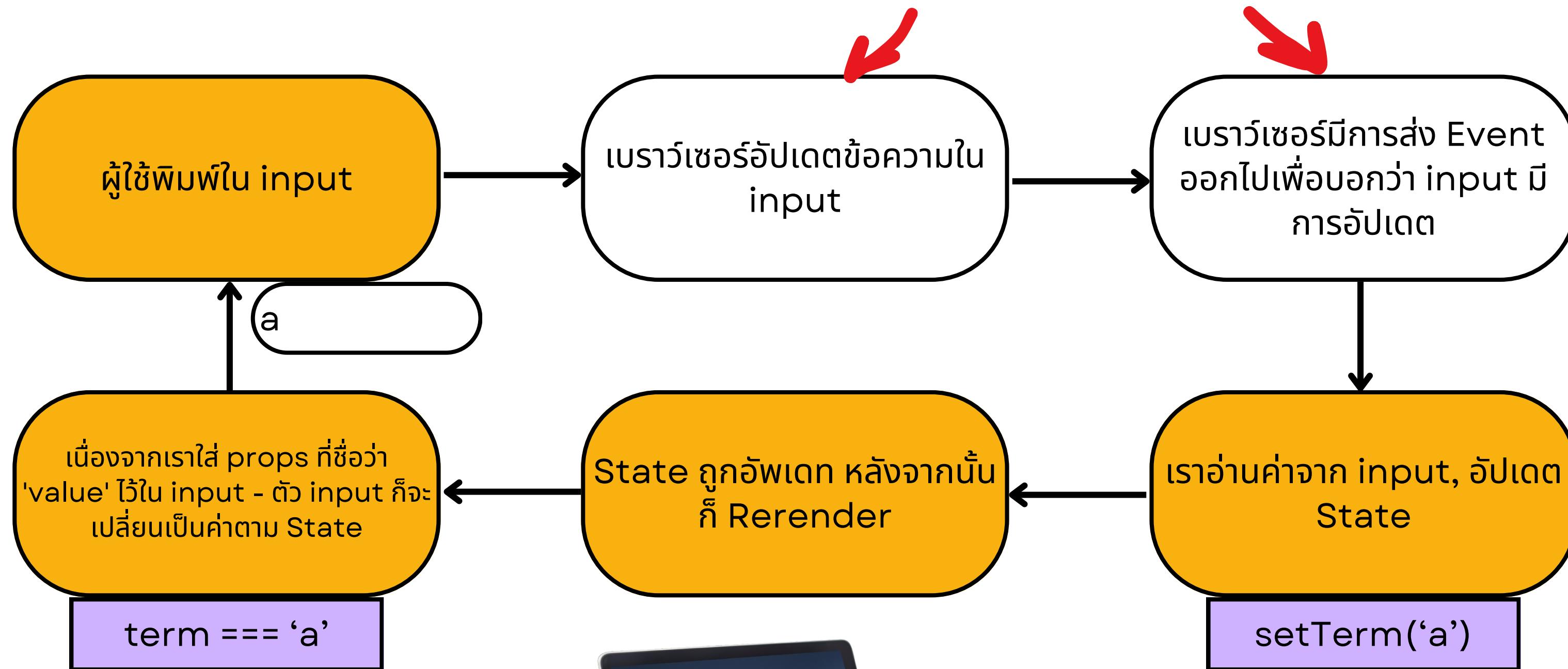
# เบราว์เซอร์เป็นคนจัดการ



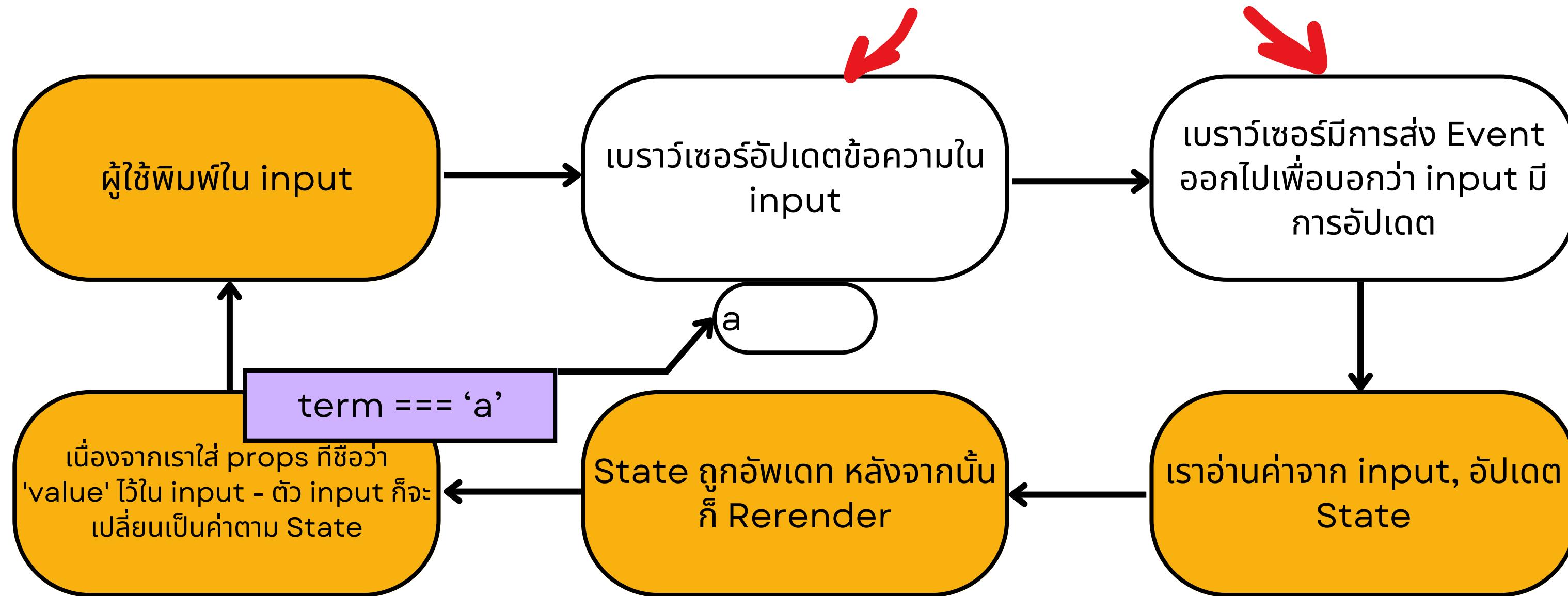
# เบราว์เซอร์เป็นคนจัดการ



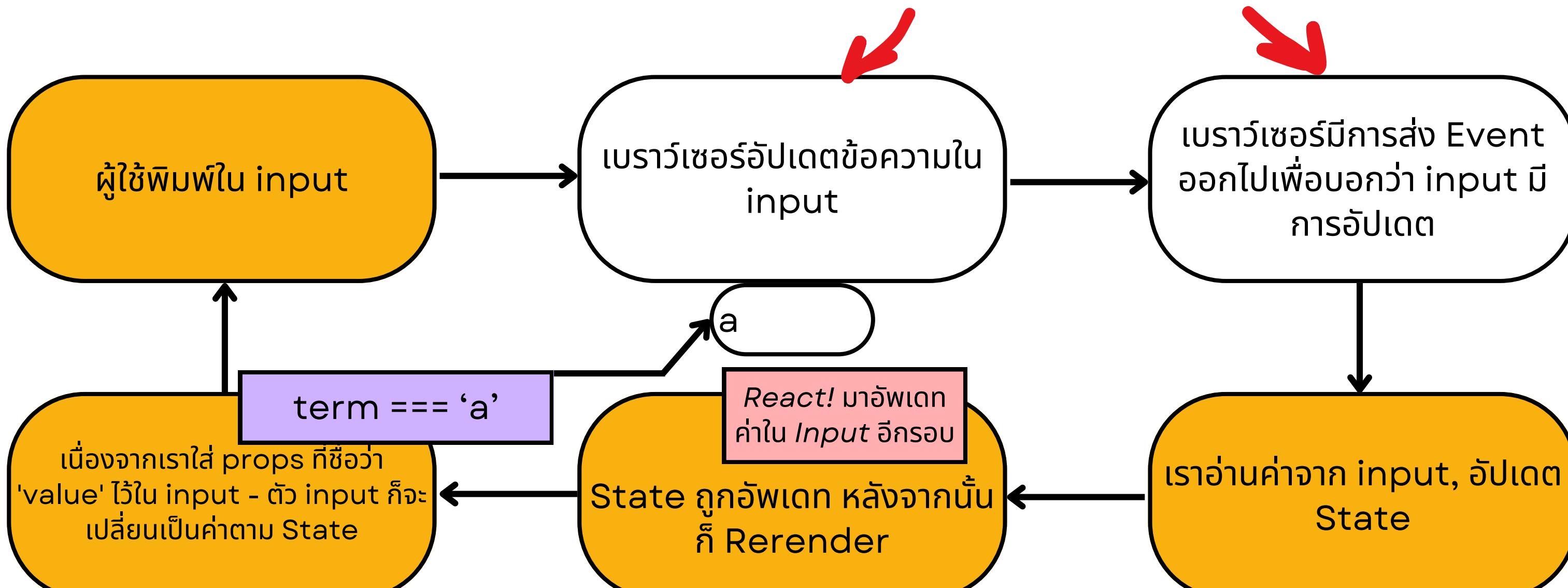
# เบราว์เซอร์เป็นคนจัดการ



# เบราว์เซอร์เป็นคนจัดการ



# เบราว์เซอร์เป็นคนจัดการ

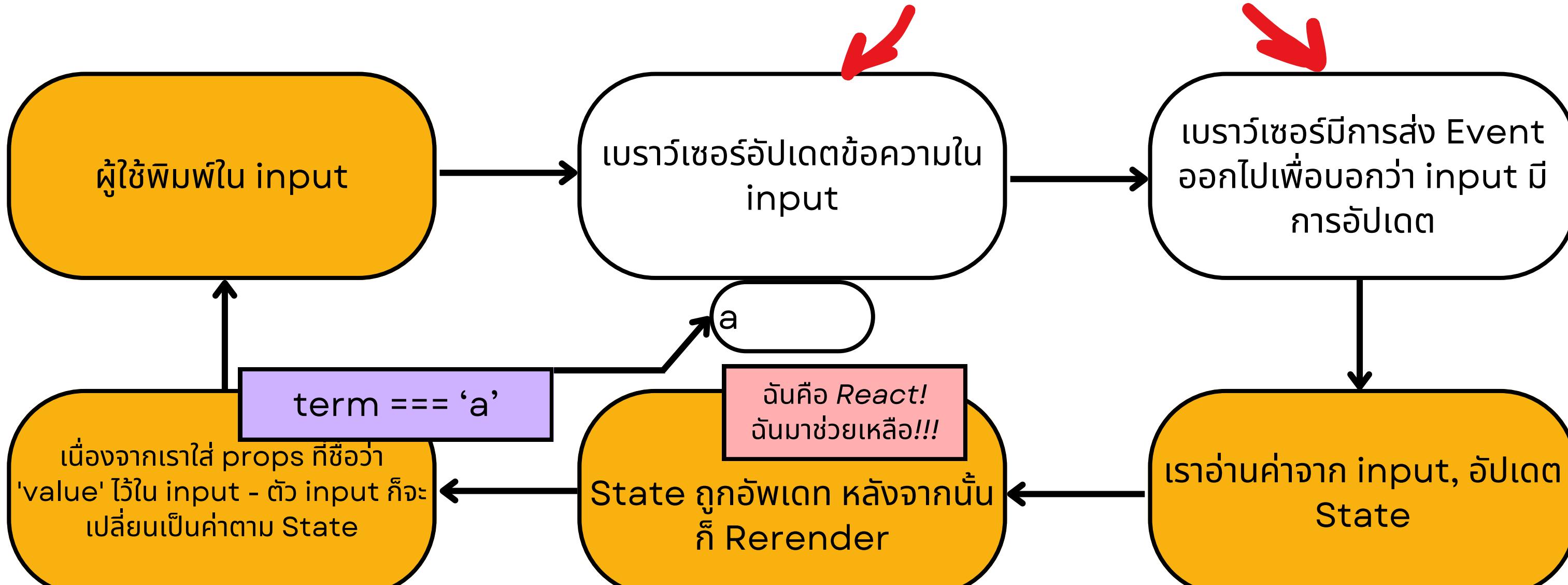




# คำเตือน!

วิธีที่ React จัดการองค์ประกอบฟอร์ม (ช่อง  
ข้อความ, ช่องกำเครื่องหมาย, ปุ่มวิทยุ ฯลฯ) มี  
ความเปลกประหลาดเล็กน้อย

# เบราว์เซอร์เป็นคนจัดการ

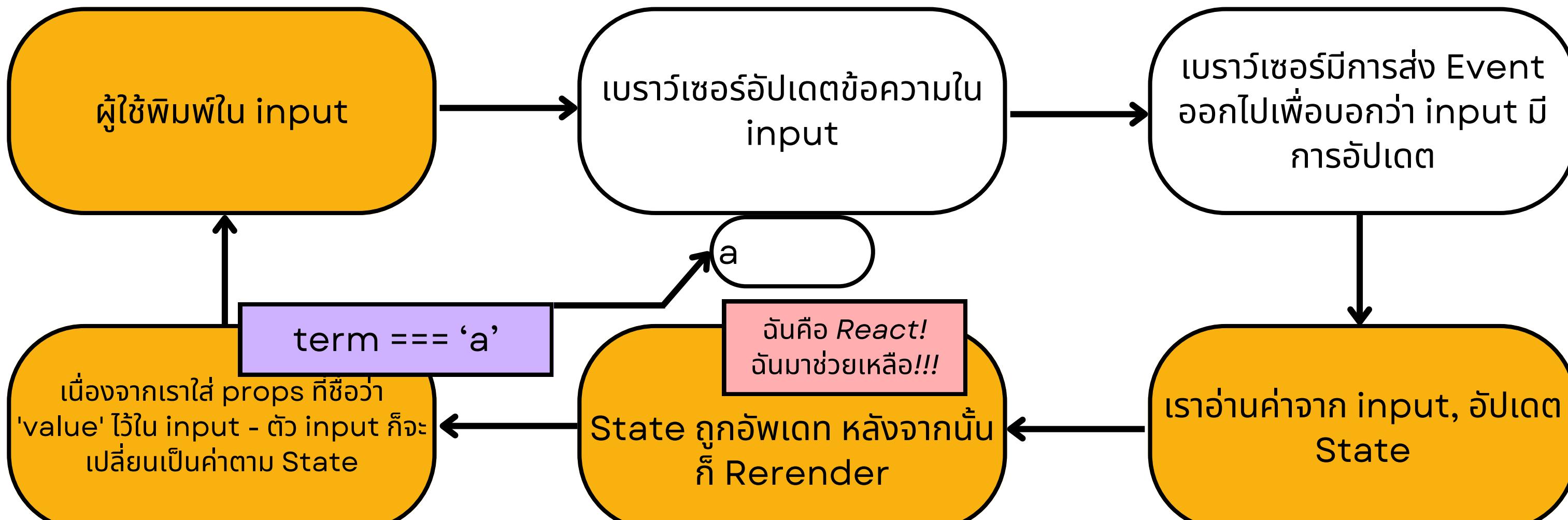




## คำเตือน!

วิธีที่ React จัดการองค์ประกอบฟอร์ม (ช่องข้อความ, ช่องกำเครื่องหมาย, ปุ่มวิทยุ ฯลฯ) มีความเปลกประหลาดเล็กน้อย

# ปฏิบัติการตามปกติของเบราว์เซอร์



# การจัดการข้อความ Inputs

1. สร้าง State ขึ้นมาใหม่

2. สร้าง event handler เพื่อสำหรับดัก Event  
'onChange'

3. ในการดึงค่าจาก input ใช้ Event ที่ได้จาก  
'onChange'

4. นำค่านั้นมาอัปเดตใน State

5. ส่ง State ไปยัง input ใน property ที่ชื่อว่า  
'value'

# ทำไมต้องทำแบบนี้???

1. ถ้าต้องการอ่านค่าของ input ดูที่ state ('term')

2. ถ้าต้องการอัปเดตค่าของ input ให้ใช้  
'setTerm('Hello')'

คอมโพเนนต์จะเรนเดอร์ใหม่ทุกครั้งที่เรามีการพิมพ์ - จัดการหลาย ๆ อย่างง่ายขึ้น



```
function SearchBar({ onSubmit }) {
  const handleFormSubmit = (event) => {
    event.preventDefault();

    onSubmit(document.querySelector('input').value);
  };

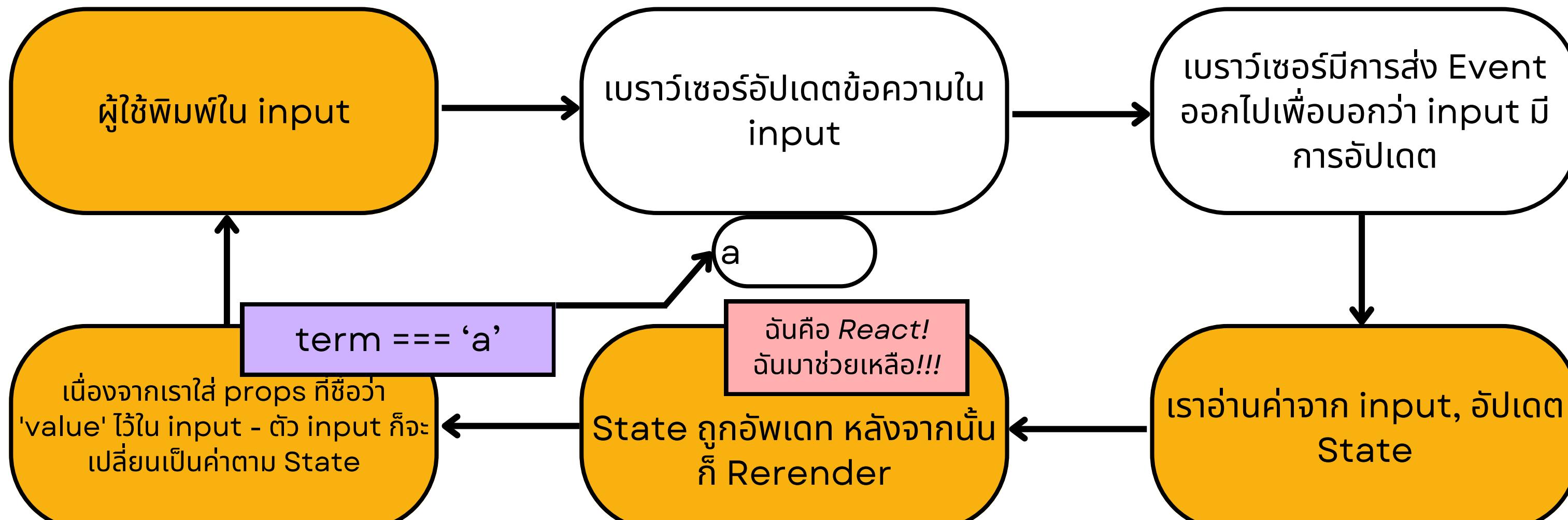
  return (
    <div>
      <form onSubmit={handleFormSubmit}>
        <input />
      </form>
    </div>
  );
}
```

อย่าทำแบบนี้  
เด็ดขาด

อย่าพยายามดึงค่าจาก input โดยใช้  
query selector (หรือวิธีที่คล้ายกัน  
ที่ใช้ Document Object)

คุณจะไม่ผ่านการสัมภาษณ์งาน  
ถ้าคุณเขียนโค้ดแบบนี้

# ปฏิบัติการตามปกติของเบราว์เซอร์



# การจัดการข้อความ Inputs

สร้างสถานะใหม่

สร้าง event handler เพื่อจับการเปลี่ยนแปลงจาก  
เหตุการณ์ 'onChange'

เมื่อเหตุการณ์ 'onChange' เกิดขึ้น ดึงค่าจาก  
input

นำค่านั้นมาจัด input และใช้มันเพื่ออัปเดตสถานะของคุณ

ส่งสถานะของคุณไปยัง input เป็นค่า property  
'value'

```
function SearchBar({ onSubmit }) {  
  const handleFormSubmit = (event) => {  
    event.preventDefault();  
  
    onSubmit(document.querySelector('input').value);  
  };  
  
  return (  
    <div>  
      <form onSubmit={handleFormSubmit}>  
        <input />  
      </form>  
    </div>  
  );  
}
```

อย่าทำแบบนี้  
เด็ดขาด

อย่าพยายามดึงค่าจาก input โดยใช้  
query selector (หรือวิธีที่คล้ายกัน)

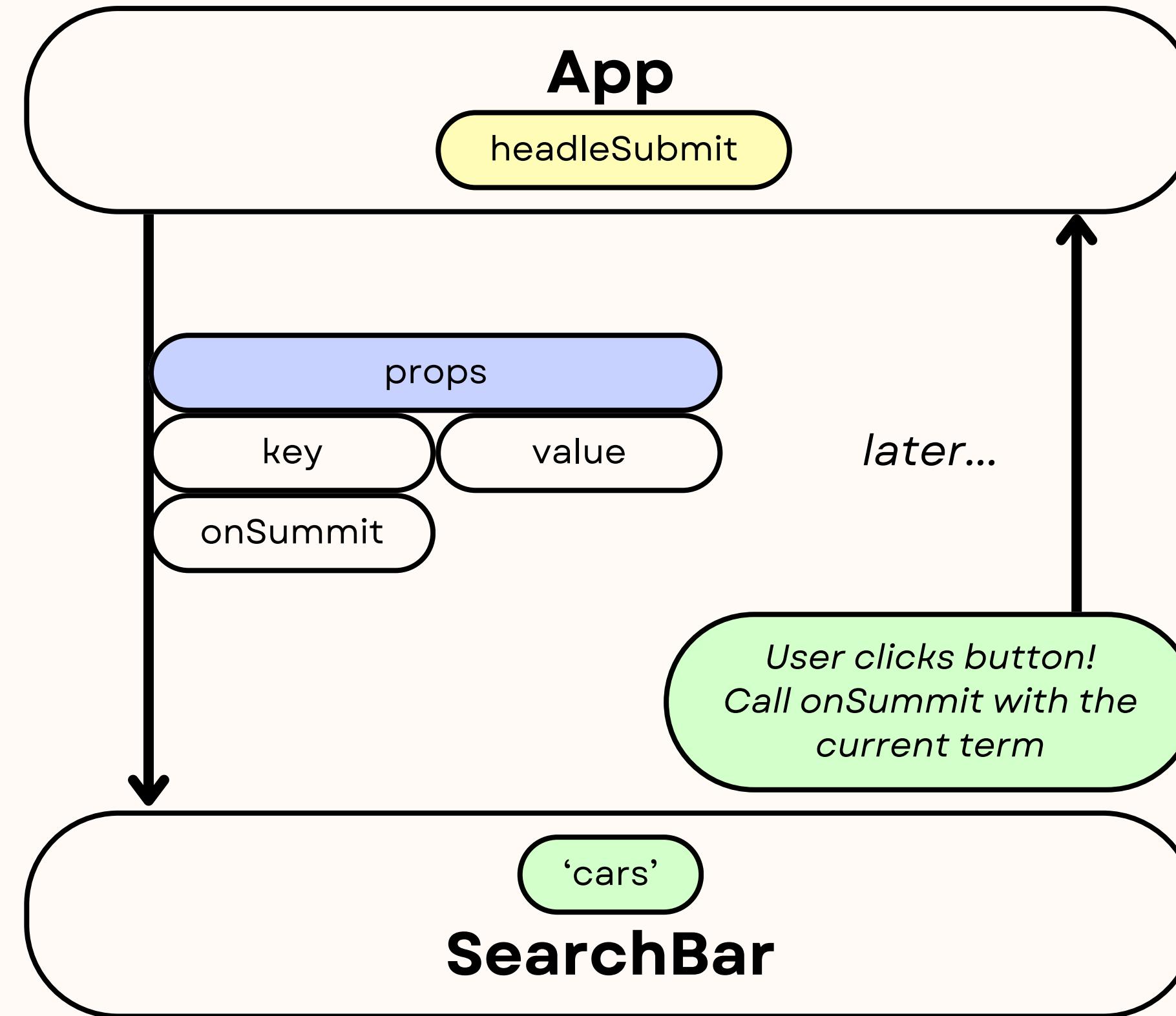
คุณจะถูกปฏิเสธจากการ  
สัมภาษณ์งานถ้าคุณเขียนโค้ด  
แบบนี้

สื่อสารจาก Child ขึ้นไปยัง Parent

ปฏิบัติกับมันเหมือนเหตุการณ์ปกติ!

ส่งต่อ event handler ลงไป

เรียก handler เมื่อมีเหตุการณ์น่าสนใจเกิดขึ้น

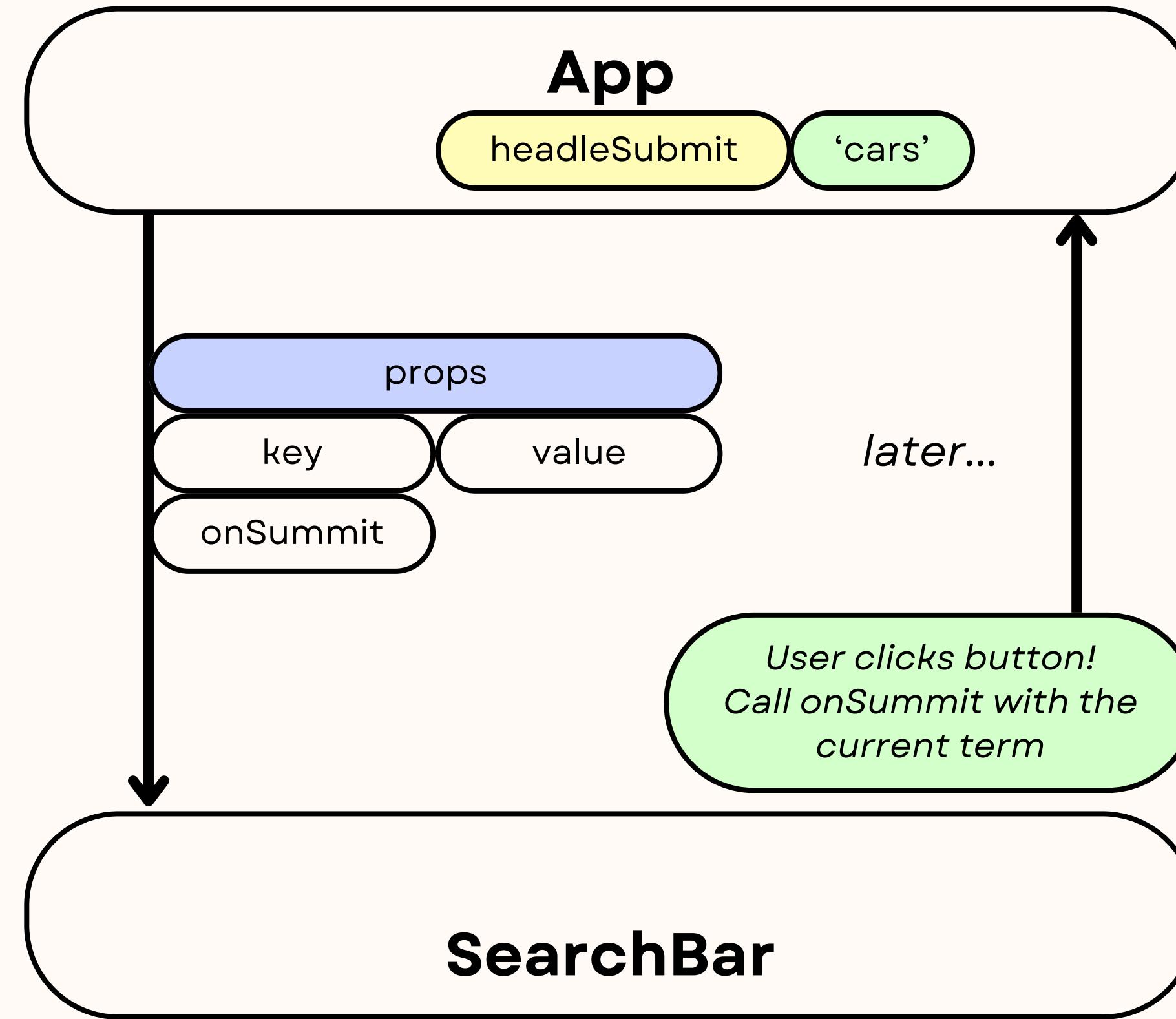


สื่อสารจาก Child ขึ้นไปยัง Parent

ปฏิบัติกับมันเหมือนเหตุการณ์ปกติ!

ส่งต่อ event handler ลงไป

เรียก handler เมื่อมีเหตุการณ์น่าสนใจเกิดขึ้น

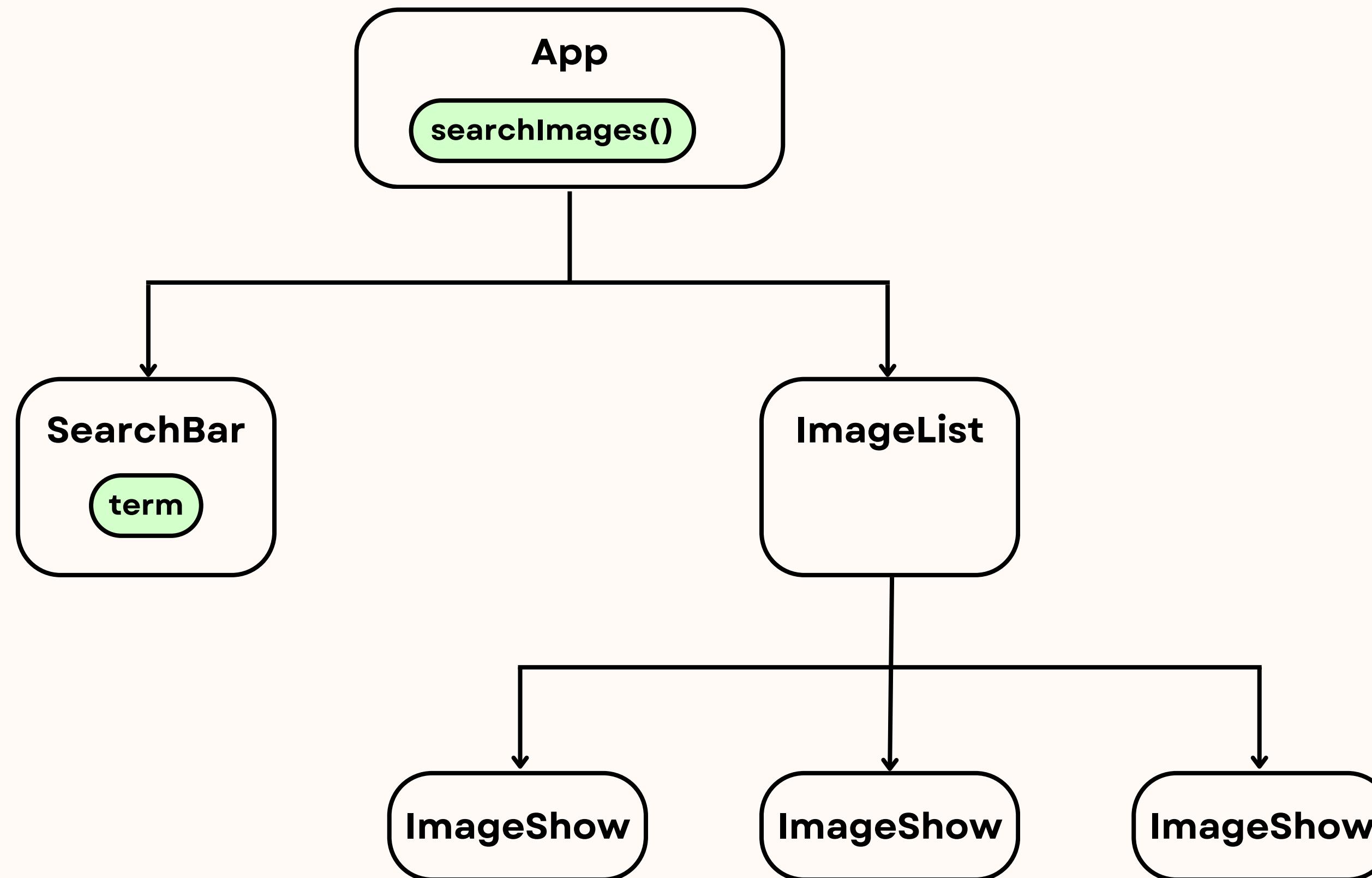


คอมโพเนนต์ SearchBar  
ประกอบด้วยช่องใส่ข้อความที่ผู้  
ใช้จะพิมพ์เข้าไป

ผู้ใช้กดปุ่ม 'enter' ในช่องใส่  
ข้อความหมายความว่าเราต้อง  
ทำการค้นหา

เรามีฟังก์ชันที่จะเปลี่ยนคำค้นหา  
เป็นอาร์เรย์ของอ็อปเจกต์ภาพ

อาร์เรย์ของอ็อปเจกต์ภาพต้อง  
ถูกส่งเข้าไปในคอมโพเนนต์  
ImageList

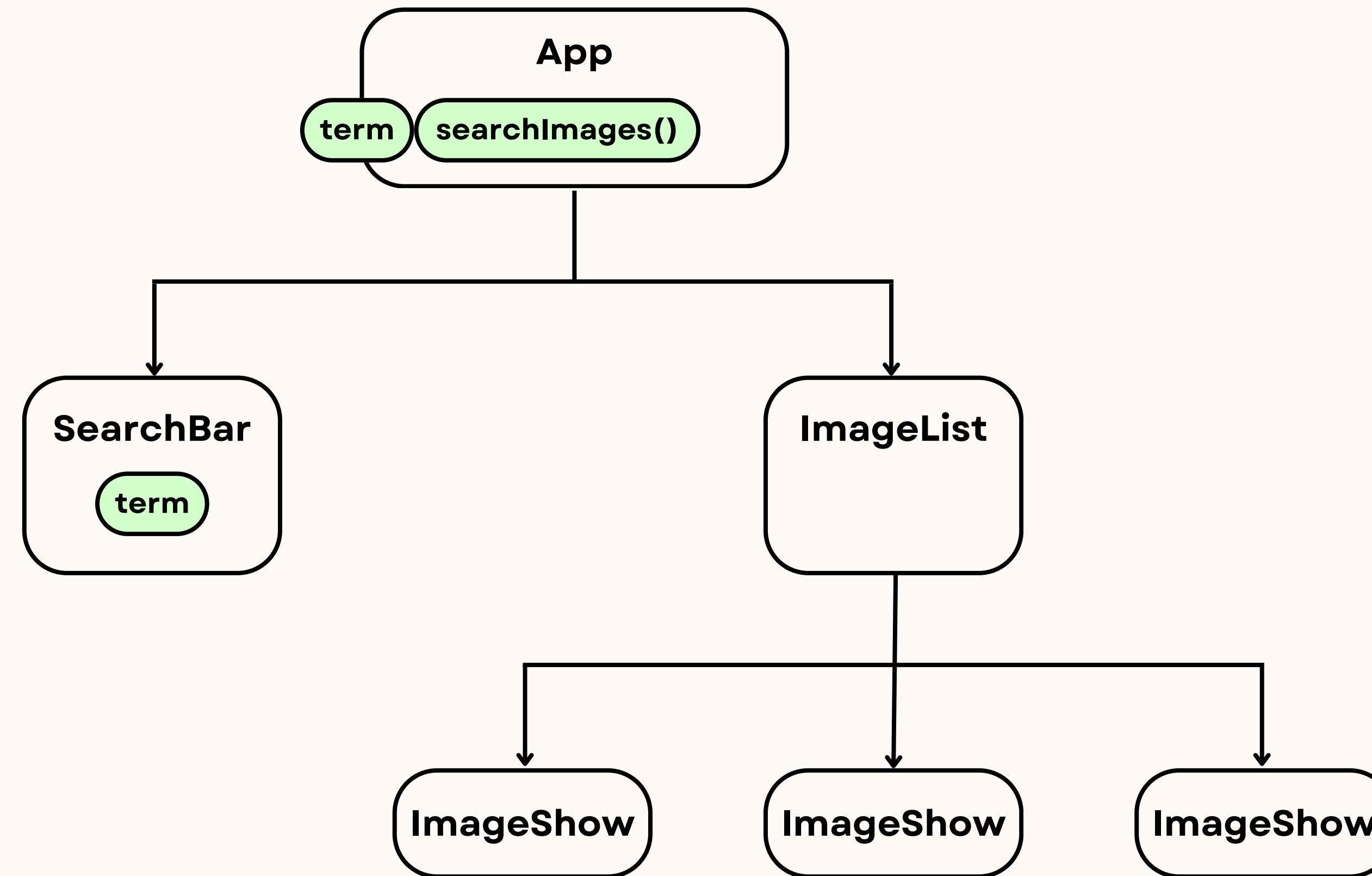


คอมโพเนนต์ SearchBar  
ประกอบด้วยช่องใส่ข้อความที่ผู้  
ใช้จะพิมพ์เข้าไป

ผู้ใช้กดปุ่ม 'enter' ในช่องใส่  
ข้อความหมายความว่าเราต้อง  
ทำการค้นหา

เรามีฟังก์ชันที่จะเปลี่ยนคำค้นหา  
เป็นอาร์เรย์ของอ็อปเจกต์ภาพ

อาร์เรย์ของอ็อปเจกต์ภาพต้อง  
ถูกส่งเข้าไปในคอมโพเนนต์  
ImageList

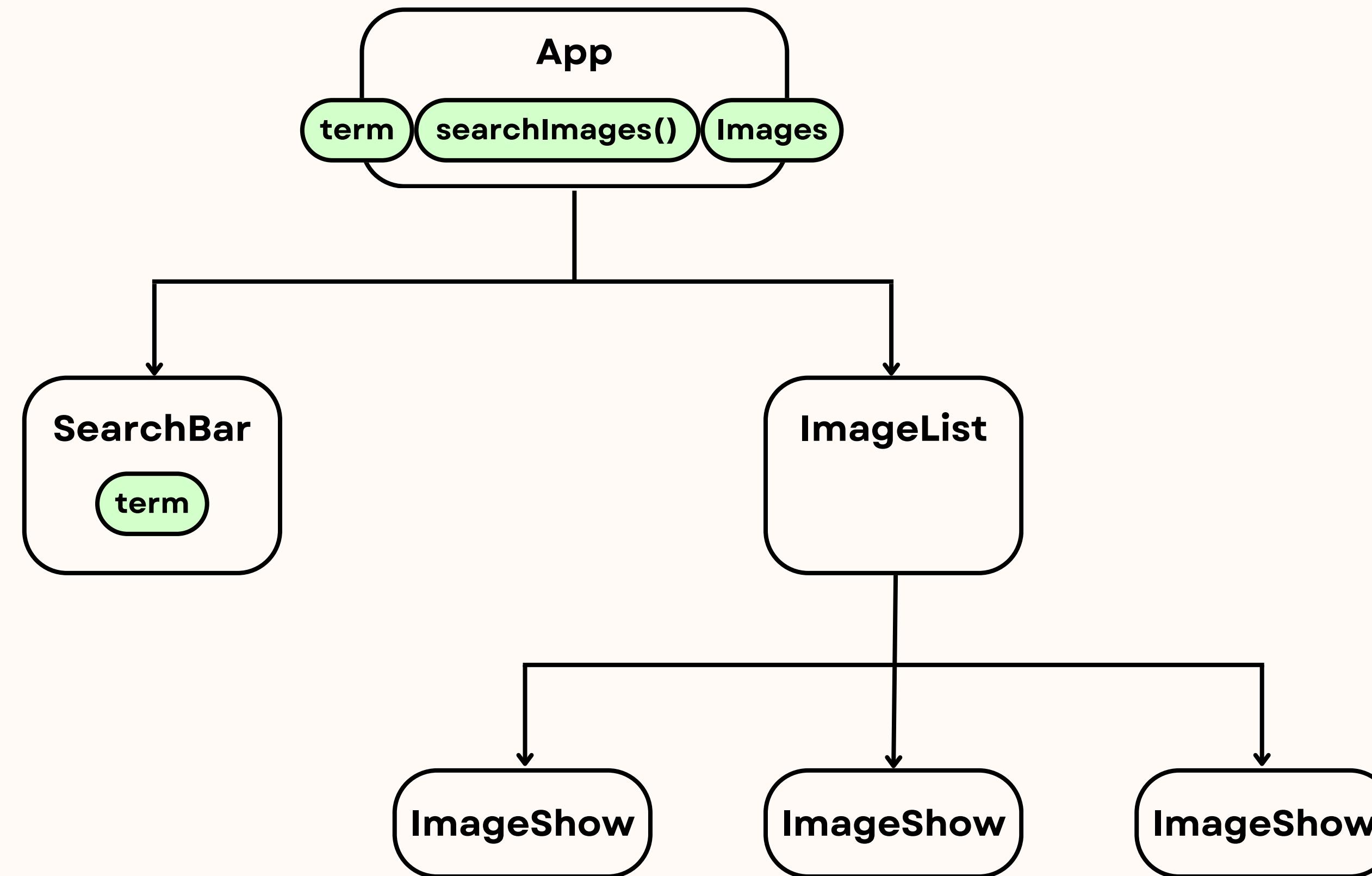


คอมโพเนนต์ SearchBar  
ประกอบด้วยช่องใส่ข้อความที่ผู้  
ใช้จะพิมพ์เข้าไป

ผู้ใช้กดปุ่ม 'enter' ในช่องใส่  
ข้อความหมายความว่าเราต้อง  
ทำการค้นหา

เรามีฟังก์ชันที่จะเปลี่ยนคำค้นหา  
เป็นอาร์เรย์ของอ็อปเจกต์ภาพ

อาร์เรย์ของอ็อปเจกต์ภาพต้อง  
ถูกส่งเข้าไปในคอมโพเนนต์  
ImageList

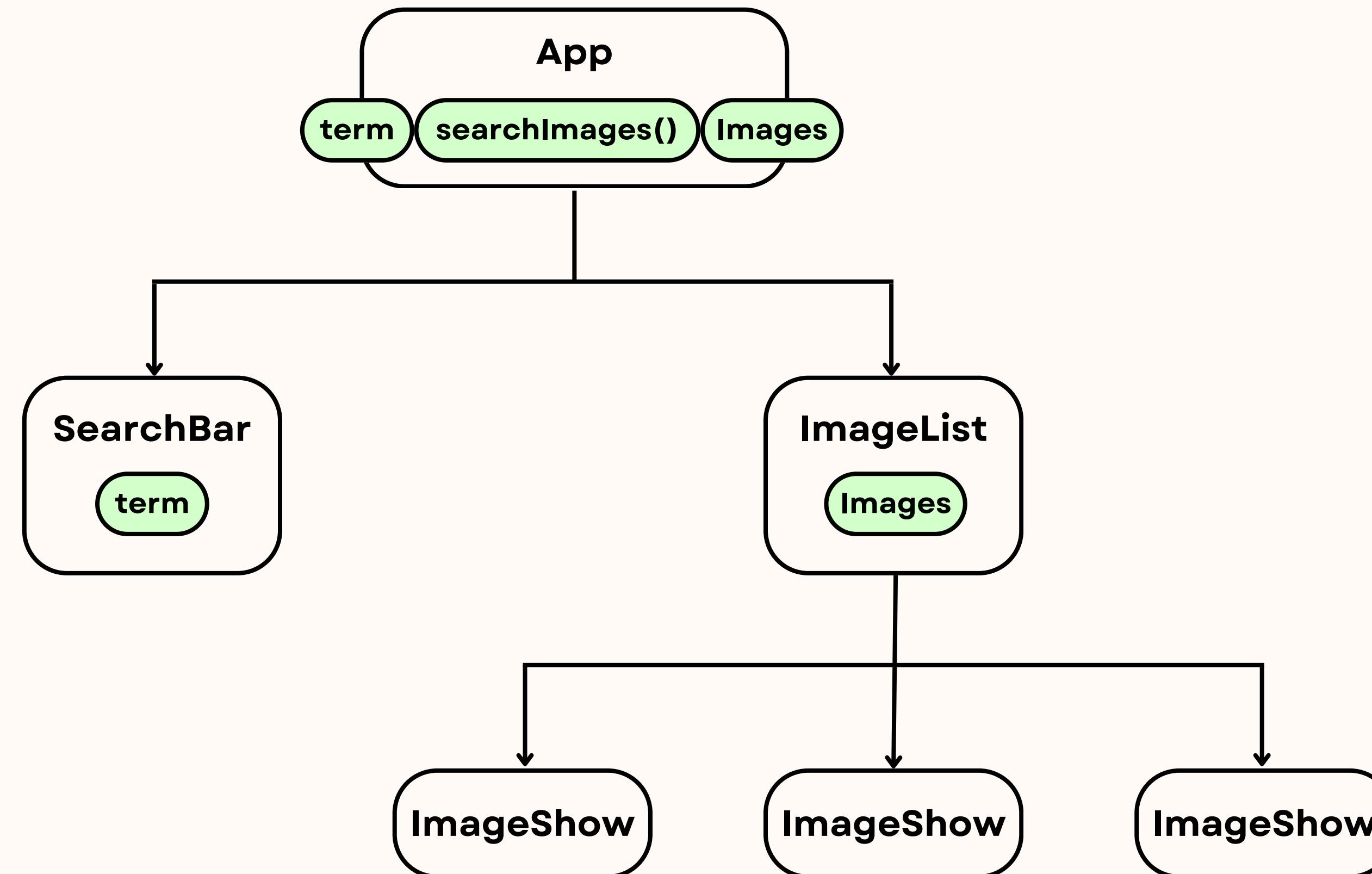


คอมโพเนนต์ SearchBar  
ประกอบด้วยช่องใส่ข้อความที่ผู้  
ใช้จะพิมพ์เข้าไป

ผู้ใช้กดปุ่ม 'enter' ในช่องใส่  
ข้อความหมายความว่าเราต้อง  
ทำการค้นหา

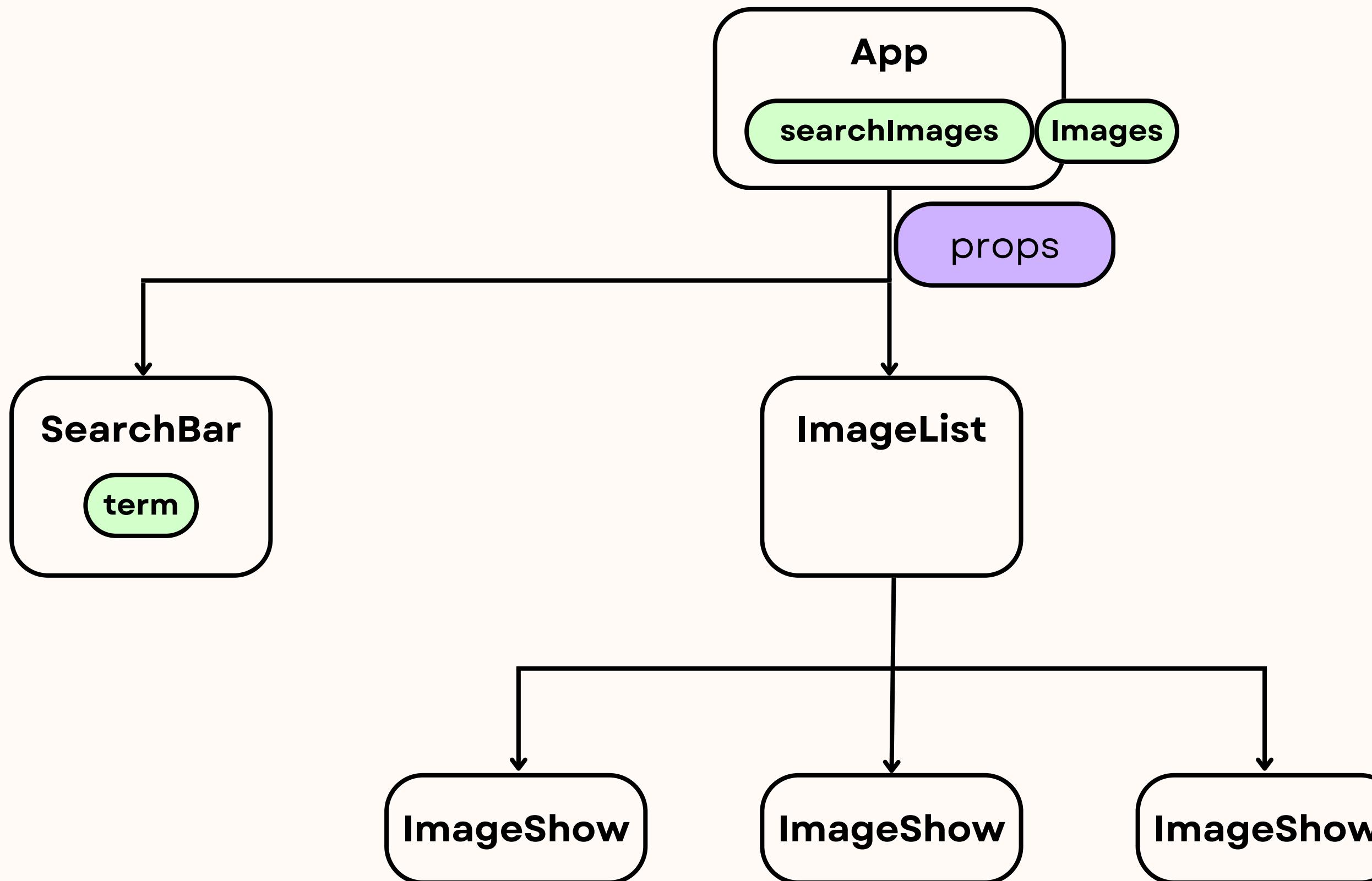
เรามีฟังก์ชันที่จะเปลี่ยนคำค้นหา  
เป็นอาร์เรย์ของอ็อปเจกต์ภาพ

อาร์เรย์ของอ็อปเจกต์ภาพต้อง  
ถูกส่งเข้าไปในคอมโพเนนต์  
ImageList



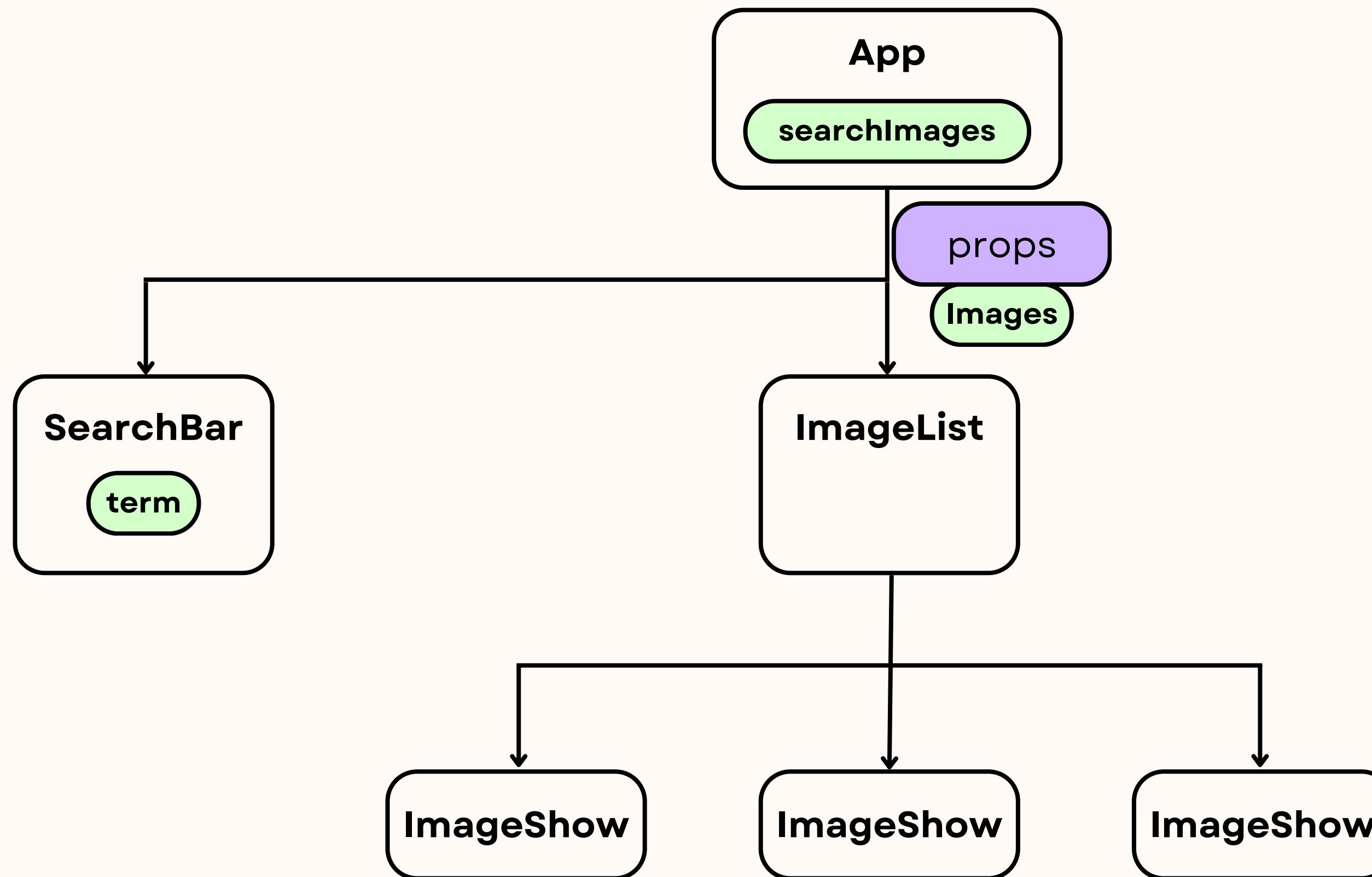
ใช้ props เพื่อสื่อสารจาก  
parent ไปยัง child

App สามารถส่งรายการของภาพไปยัง ImageList  
โดยใช้ props!



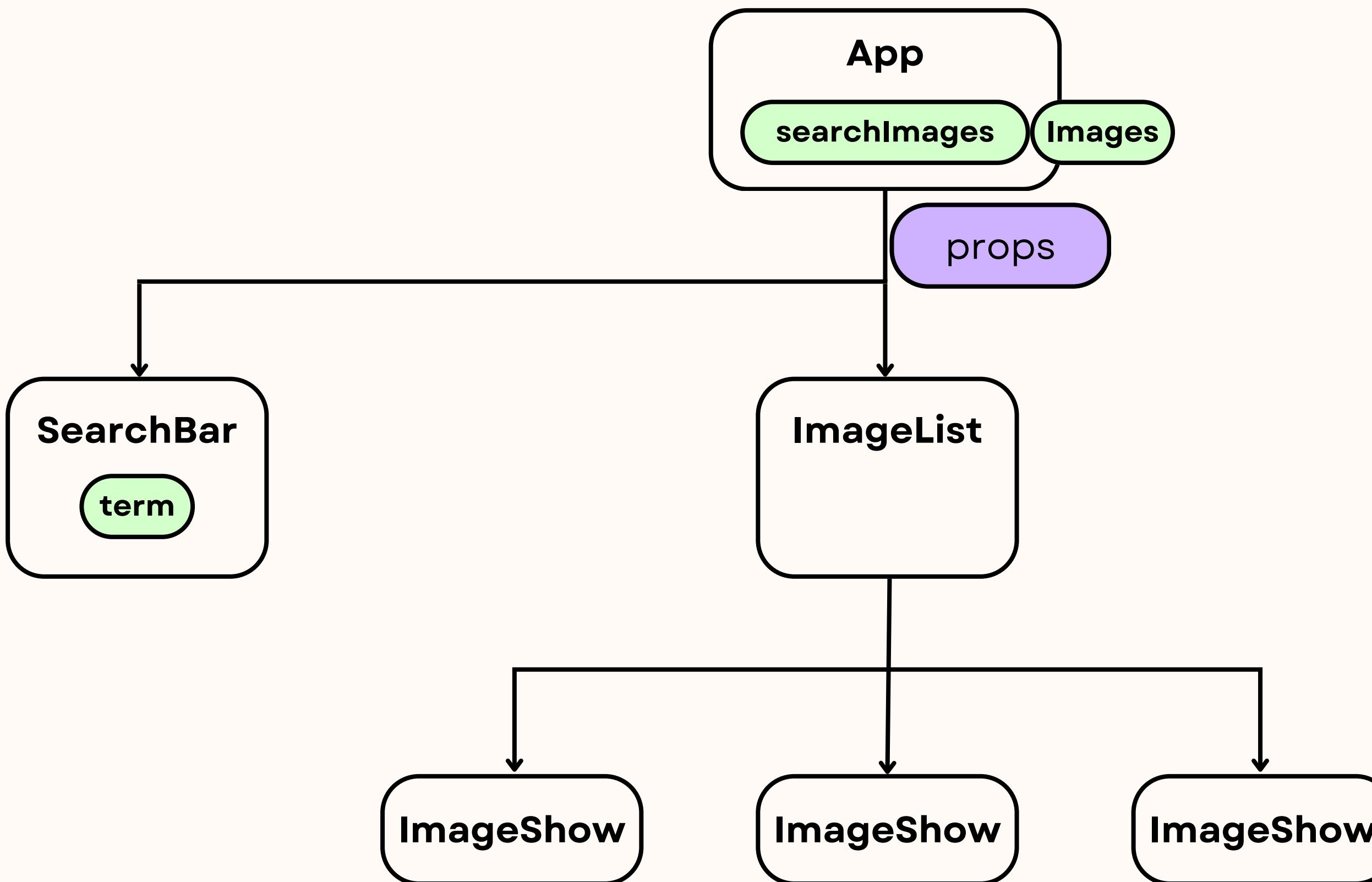
ใช้ props เพื่อสื่อสารจาก  
parent ไปยัง child

App สามารถส่งรายการของภาพไปยัง ImageList  
โดยใช้ props!



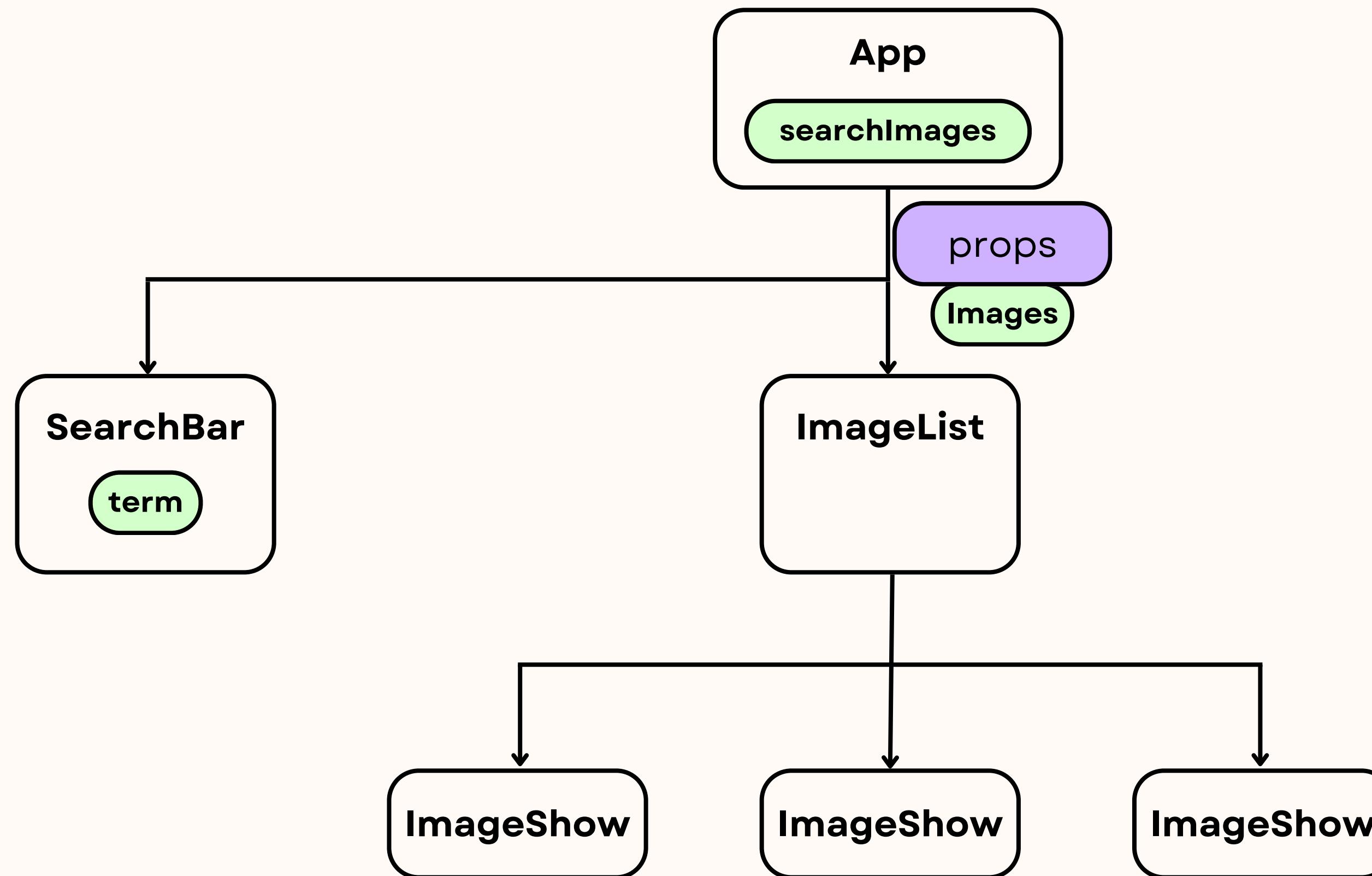
ใช้ props เพื่อสื่อสารจาก  
parent ไปยัง child

App สามารถส่งรายการของภาพไปยัง ImageList  
โดยใช้ props!



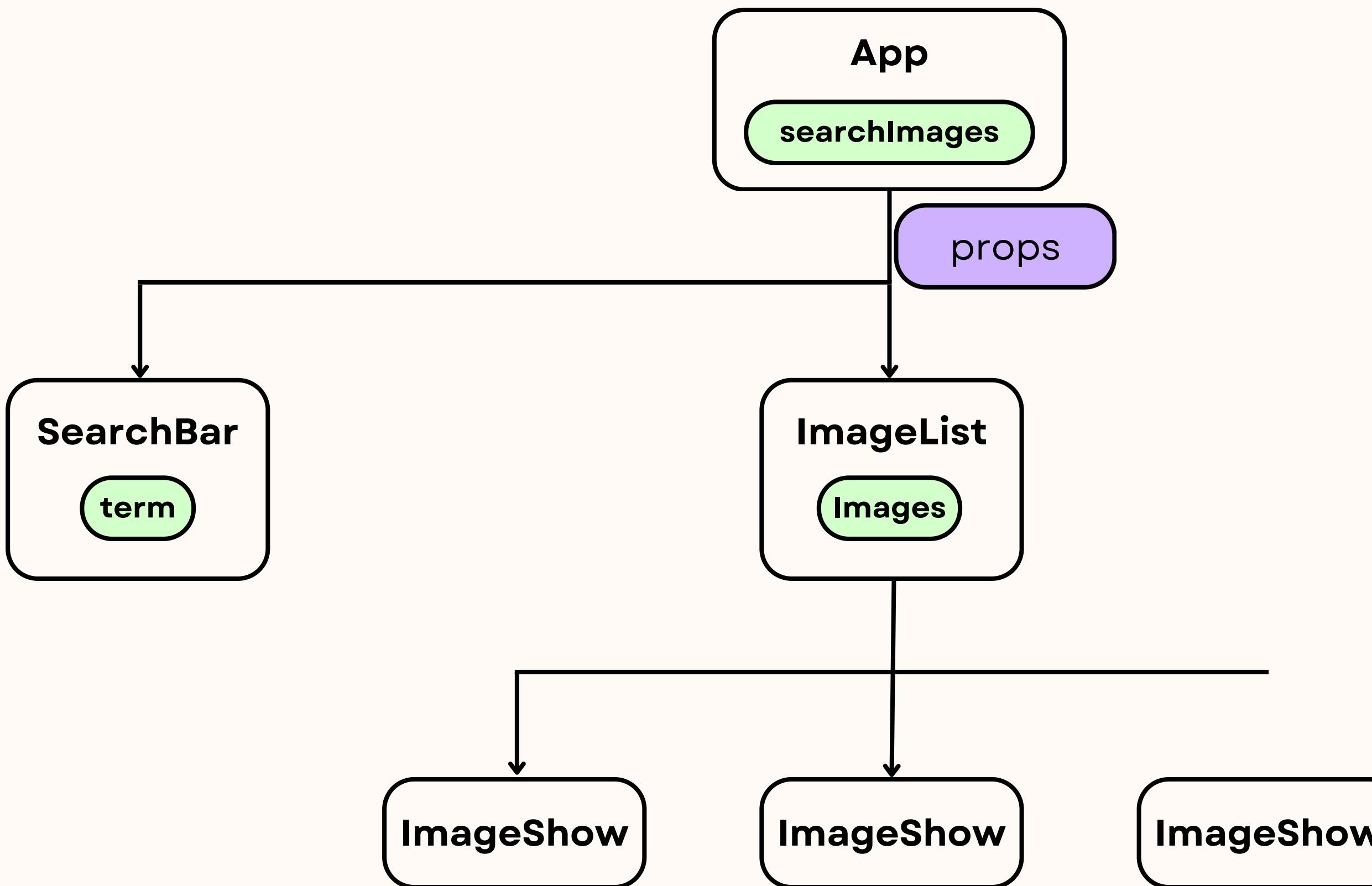
ใช้ props เพื่อสื่อสารจาก  
parent ไปยัง child

App สามารถส่งรายการของภาพไปยัง ImageList  
โดยใช้ props!



ใช้ props เพื่อสื่อสารจาก  
parent ไปยัง child

App สามารถส่งรายการของภาพไปยัง ImageList  
โดยใช้ props!



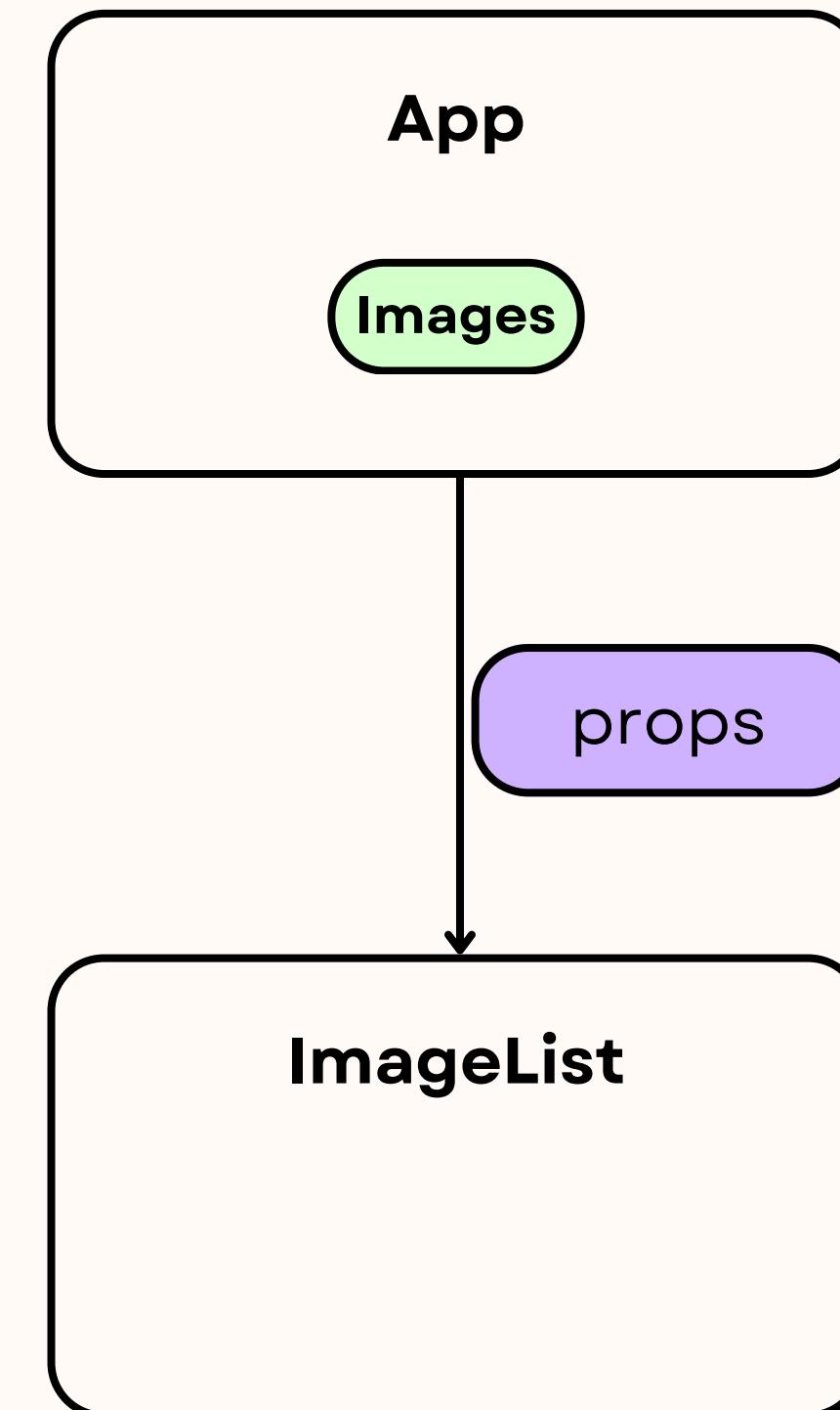


เราต้องการส่ง Array ของ Images ลงไปยัง ImageList

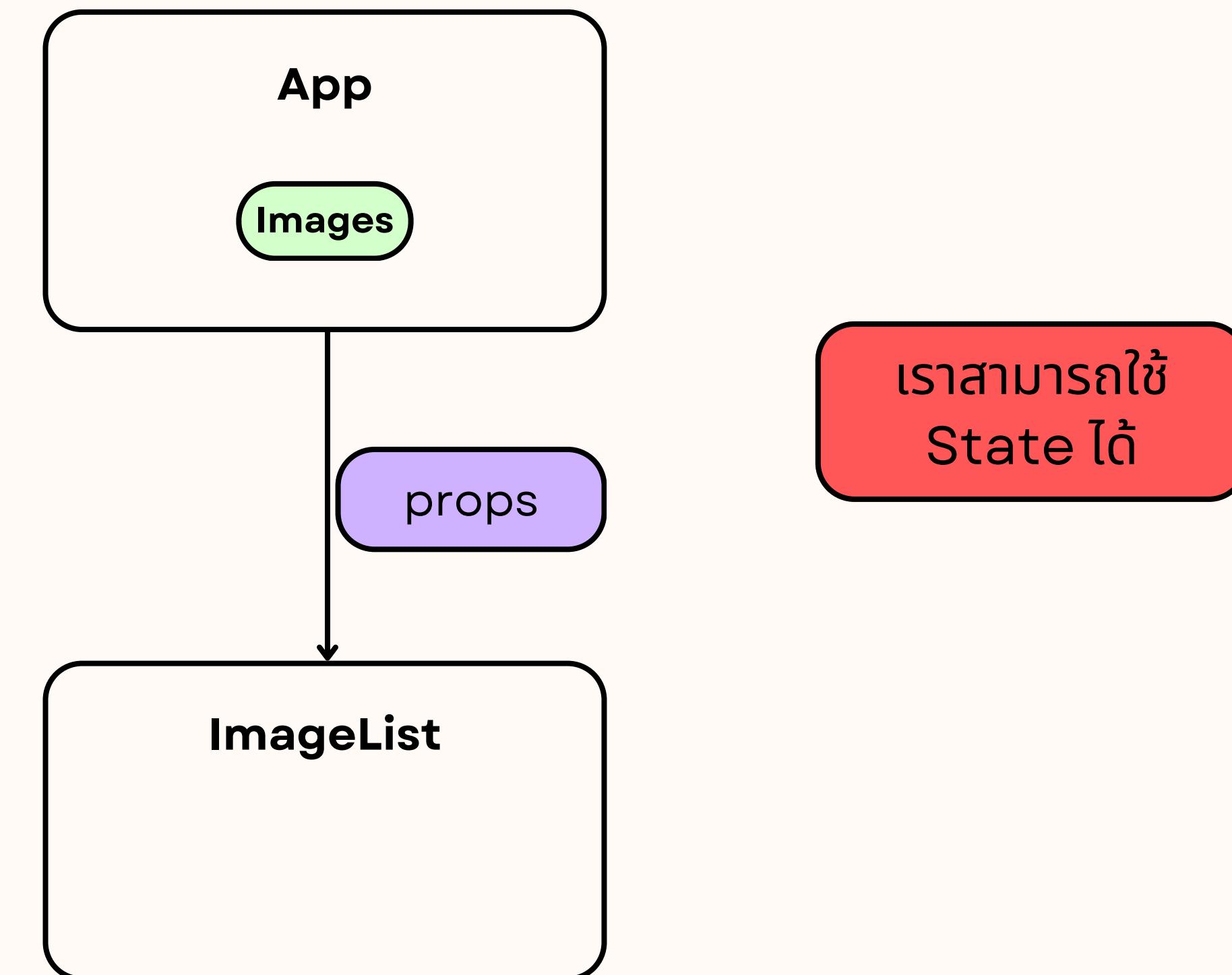
ขั้นตอนนี้อาจจงงบดหน่อย เพราะเราจำลังรวมเรื่อง state กับ props เข้าไว้ด้วยกัน

เราจะอธิบายเกี่ยวกับระบบ state เพิ่มเติม

หลังจากทำการค้นหา เราต้องการอัปเดตเนื้อหาบน  
หน้าจอด้วยรายการรูปภาพใหม่



หลังจากทำการค้นหา เราต้องการอัปเดตเนื้อหาบนหน้าจอด้วยรายการรูปภาพใหม่



```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

## Render รอบแรก

## Render รอบที่สอง

## Render รอบที่สาม

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

## Render รอบแรก

## Render รอบที่สอง

## Render รอบที่สาม

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

## Render รอบแรก

## Render รอบที่สอง

## Render รอบที่สาม

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

เรามีการเรียก  
setImages  
state มีการ  
เปลี่ยนแปลง จึง  
ต้องมีการเรนเดอร์  
ใหม่!

```
function App() {
  const [images, setImages] = useState([]);
  const handleSubmit = async (term) => {
    const result = await searchImages(term);
    setImages(result);
  }

  return (
    <div>
      <SearchBar onSubmit={handleSubmit} />
      <ImageList images={images} />
    </div>
  );
}
```

## Render รอบแรก

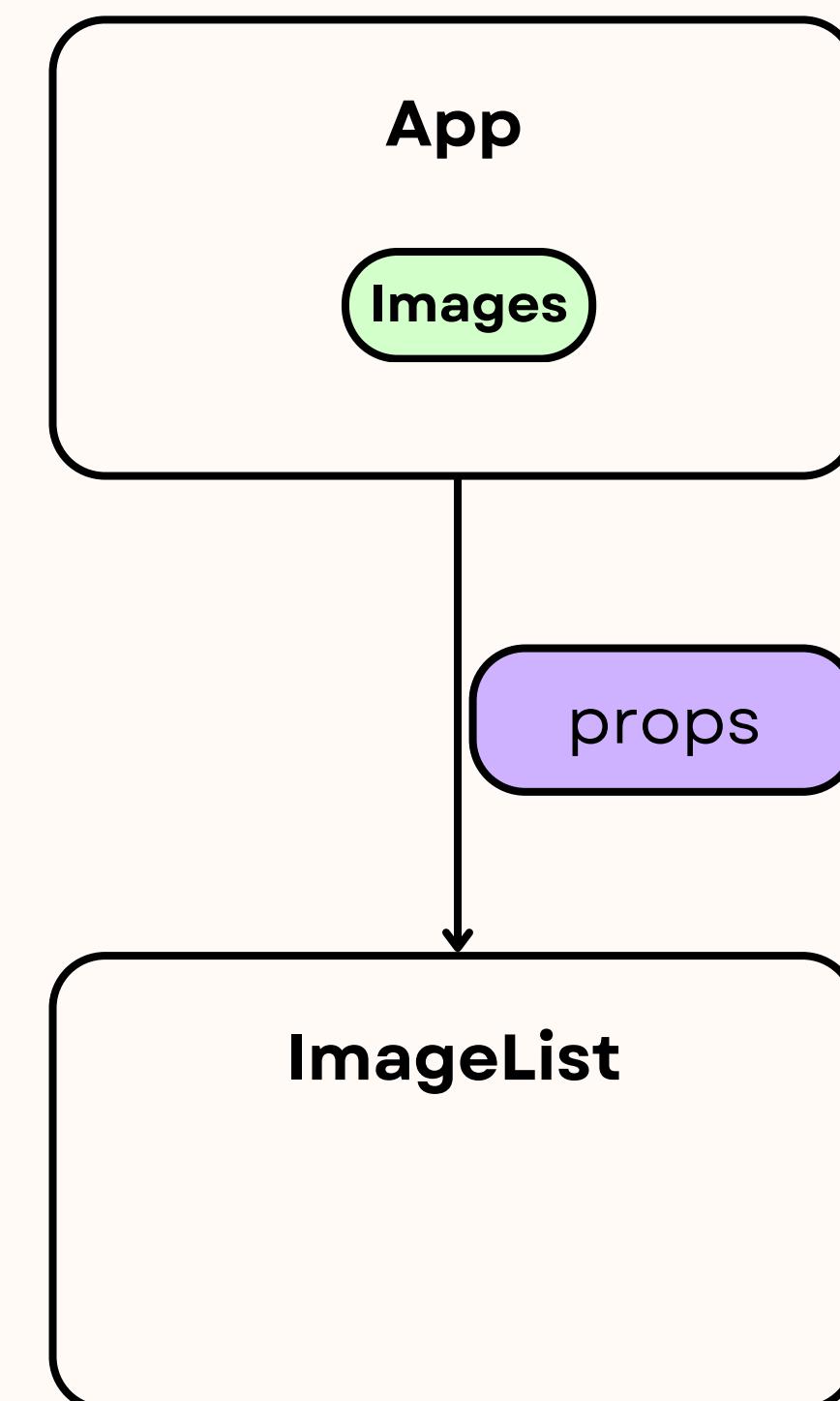
## Render รอบที่สอง

## Render รอบที่สาม

เมื่อเรามีการอัพเดต State  
Child components  
ทั้งหมดจะถูกเรนเดอร์ใหม่

จากมุมมองของ 'App',  
images เป็น state

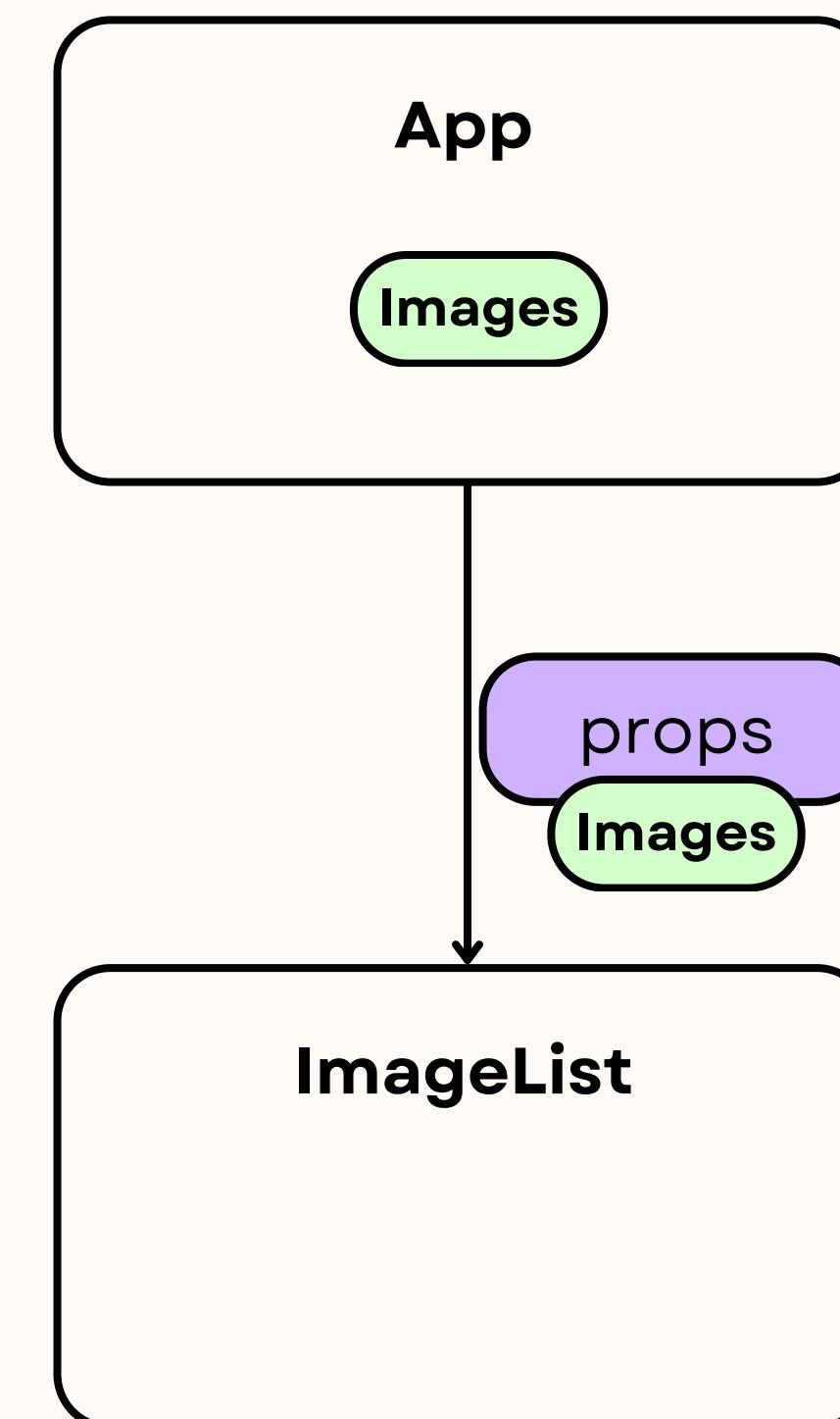
จากมุมมองของ 'ImageList',  
images เป็น props



เมื่อเรามีการอัพเดต State  
Child components  
ทั้งหมดจะถูกเรนเดอร์ใหม่

จากมุมมองของ 'App',  
images เป็น state

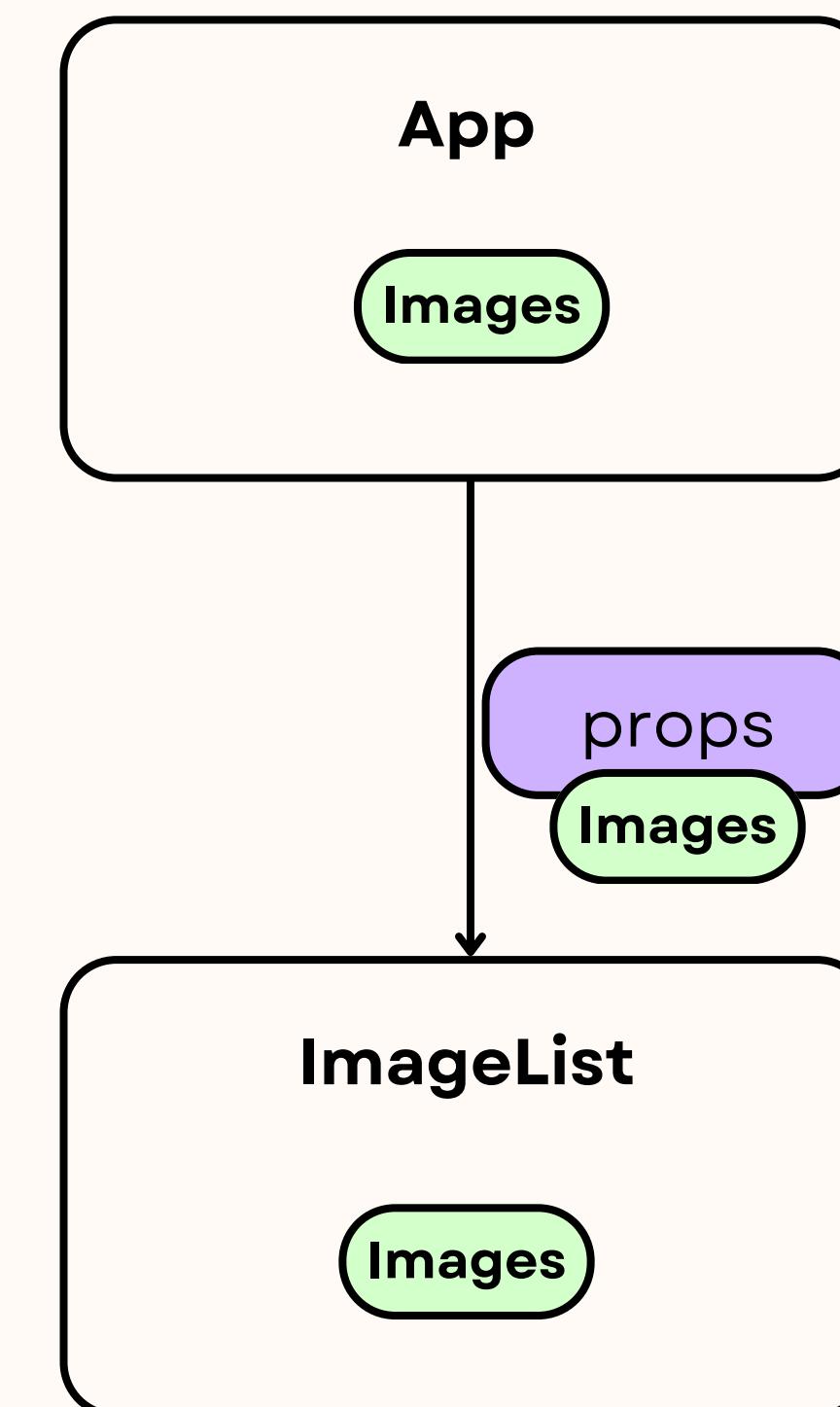
จากมุมมองของ 'ImageList',  
images เป็น props



เมื่อเรามีการอัพเดต State  
Child components  
ทั้งหมดจะถูกเรนเดอร์ใหม่

จากมุมมองของ 'App',  
images เป็น state

จากมุมมองของ 'ImageList',  
images เป็น props

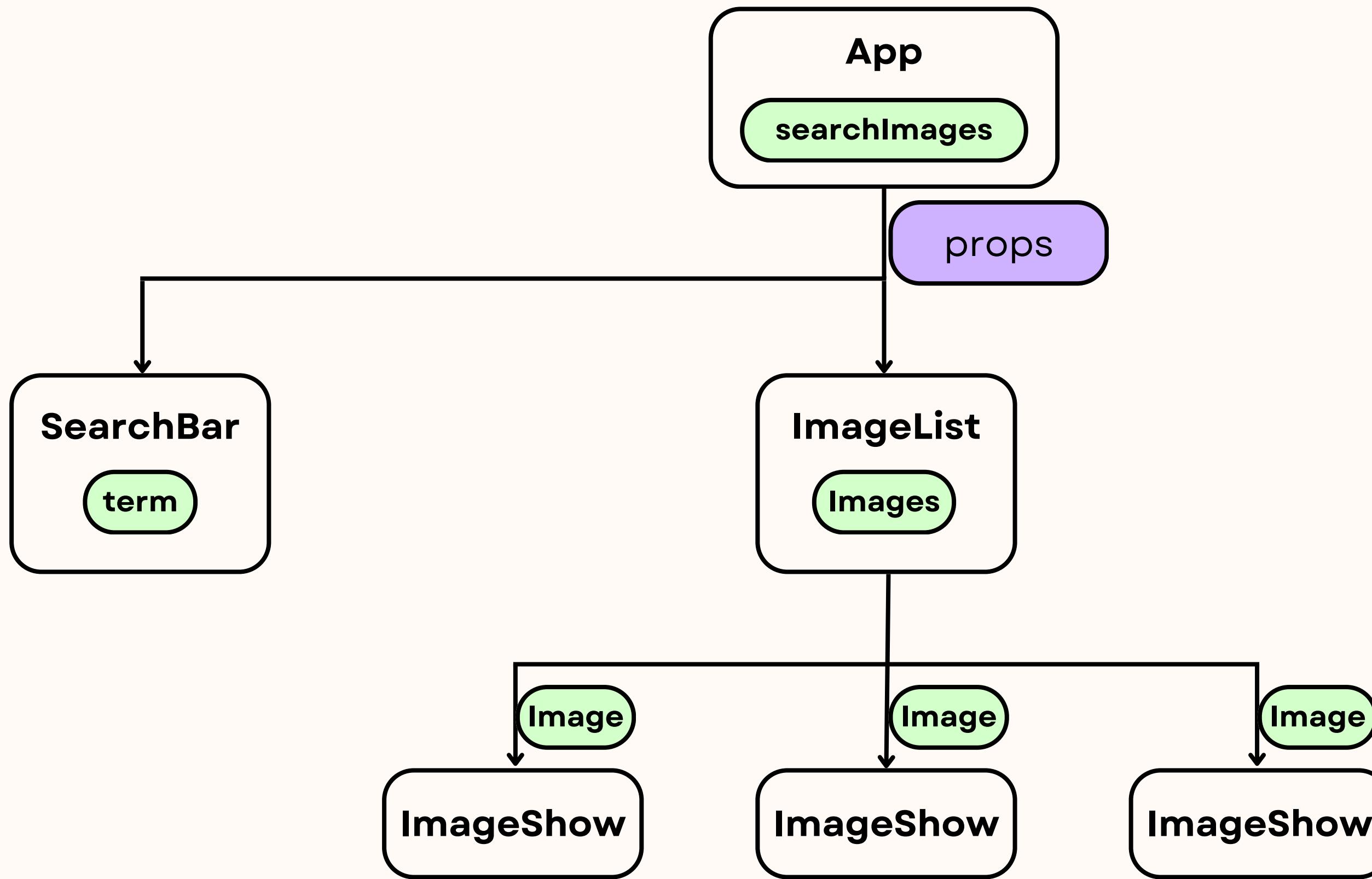




เราต้องการสื่อสารว่าเรย์ของภาพ  
ลงไปยัง ImageList

สิ่งนี้อาจดูสับสน เพราะเรากำลังจะผสมระบบสถานะ  
และระบบ props กัน

จับจังให้คำแนะนำสองประการ  
เกี่ยวกับระบบสถานะ



## images

{id: 'lk32r'}

{id: 'bk5o'}

{id: 'Ok52'}

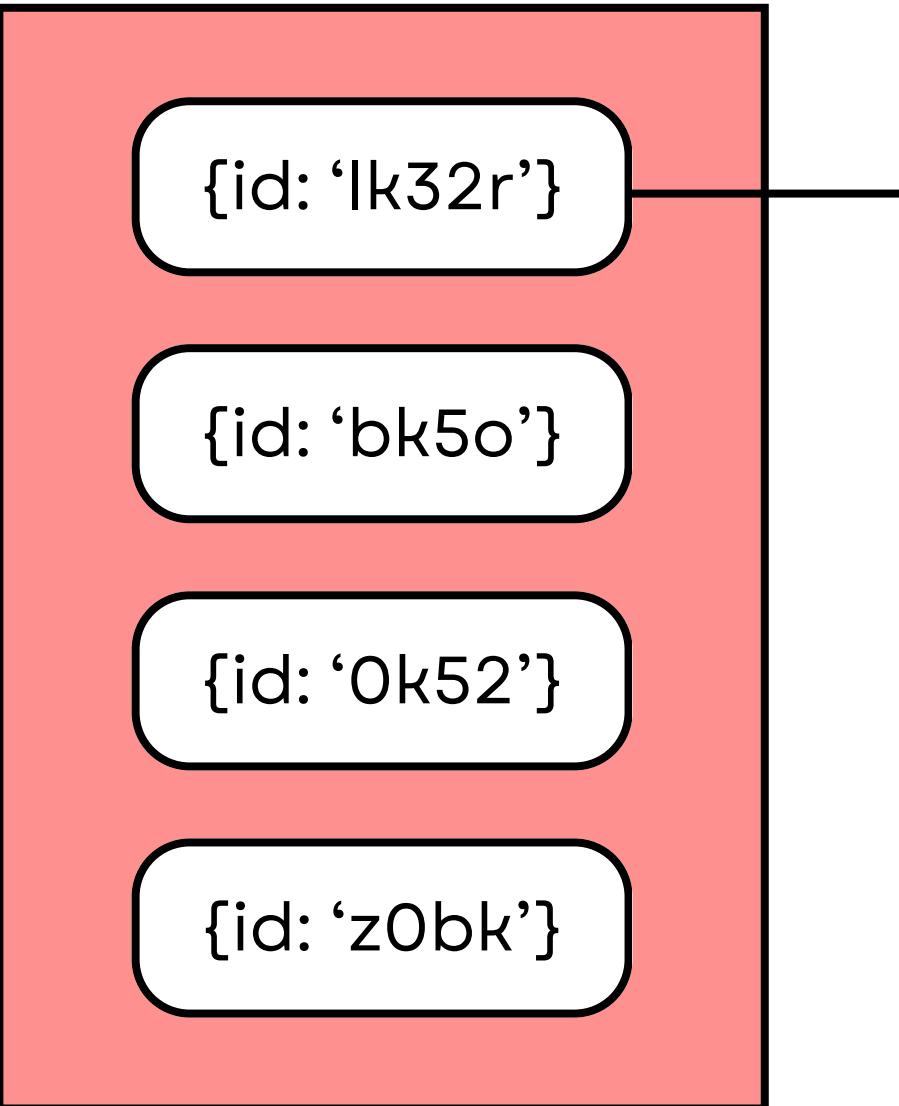
{id: 'z0bk'}

```
( image ) => {  
    return <ImageShow image={image} />  
}
```

MAPPING  
FUNCTION

## renderedImages

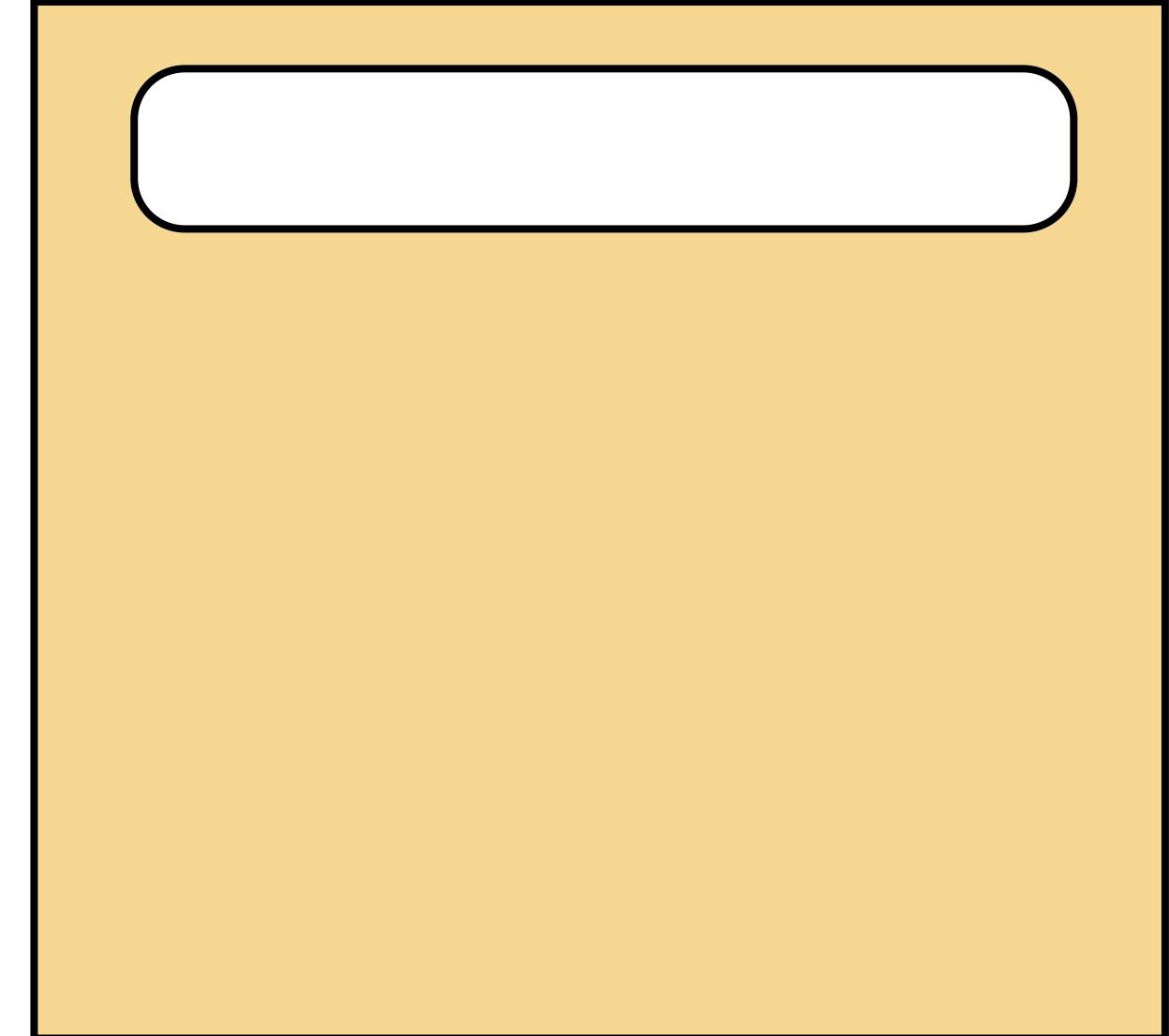
## images



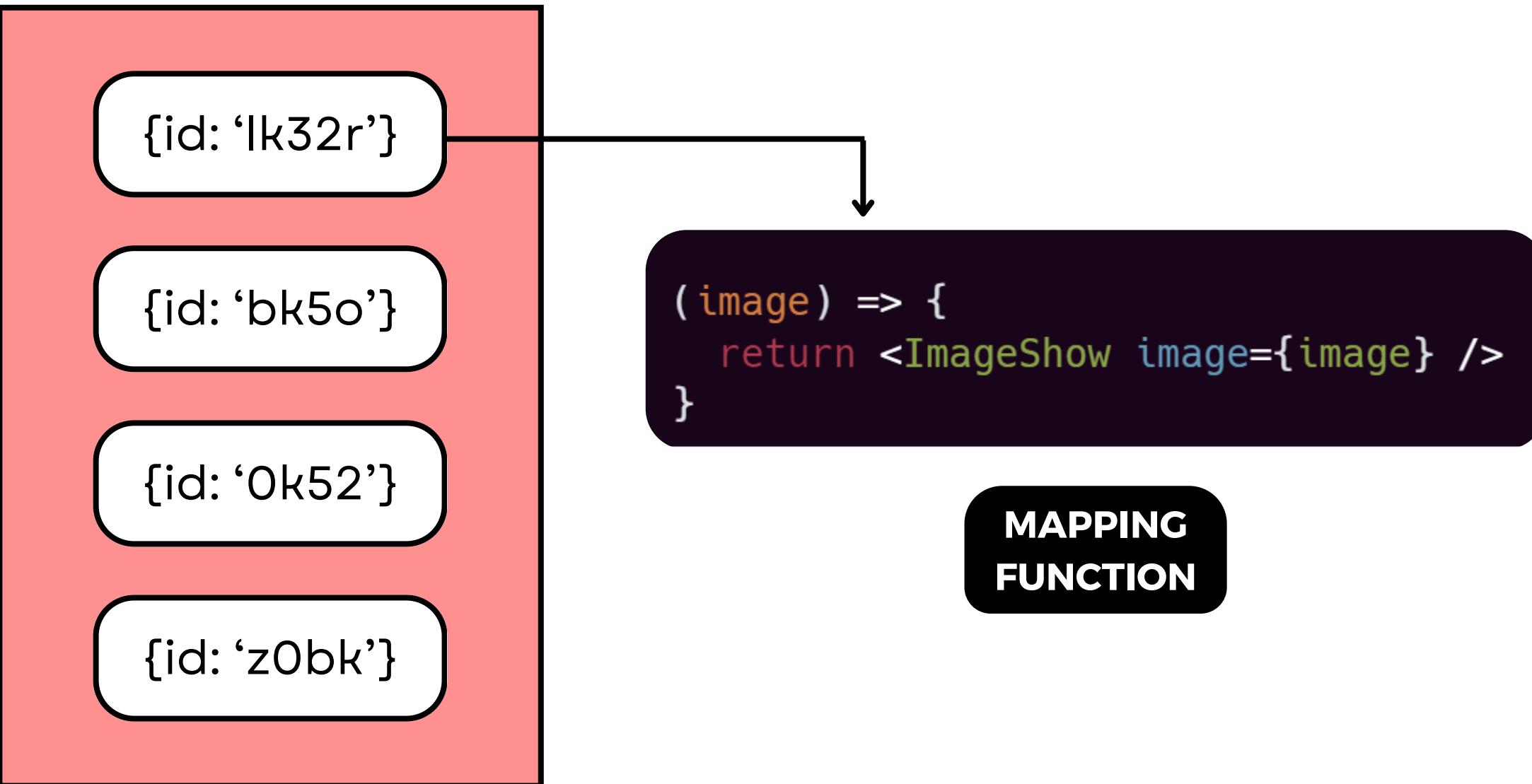
```
(image) => {  
  return <ImageShow image={image} />  
}
```

MAPPING  
FUNCTION

## renderedImages



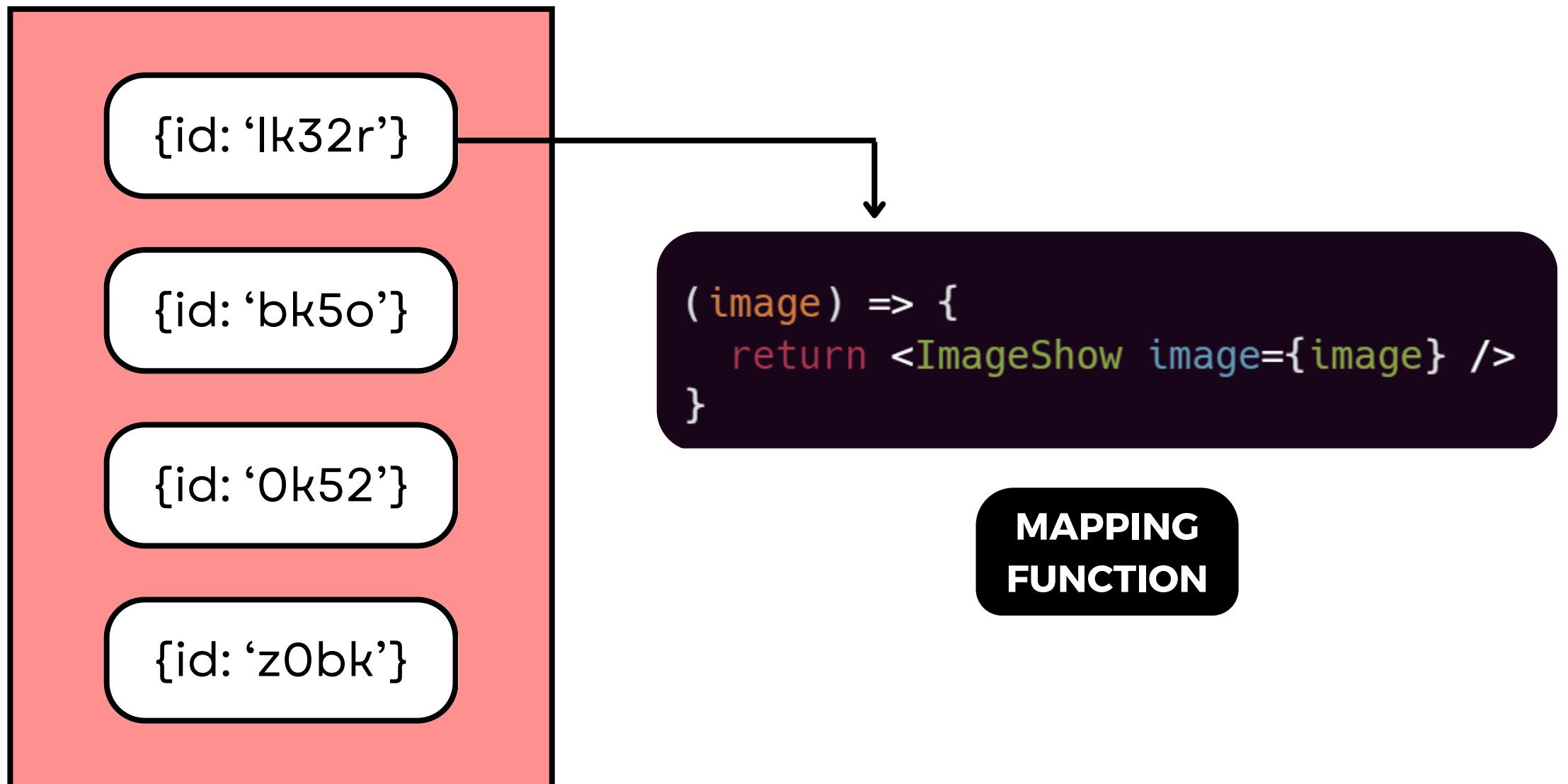
## images



## renderedImages



## images



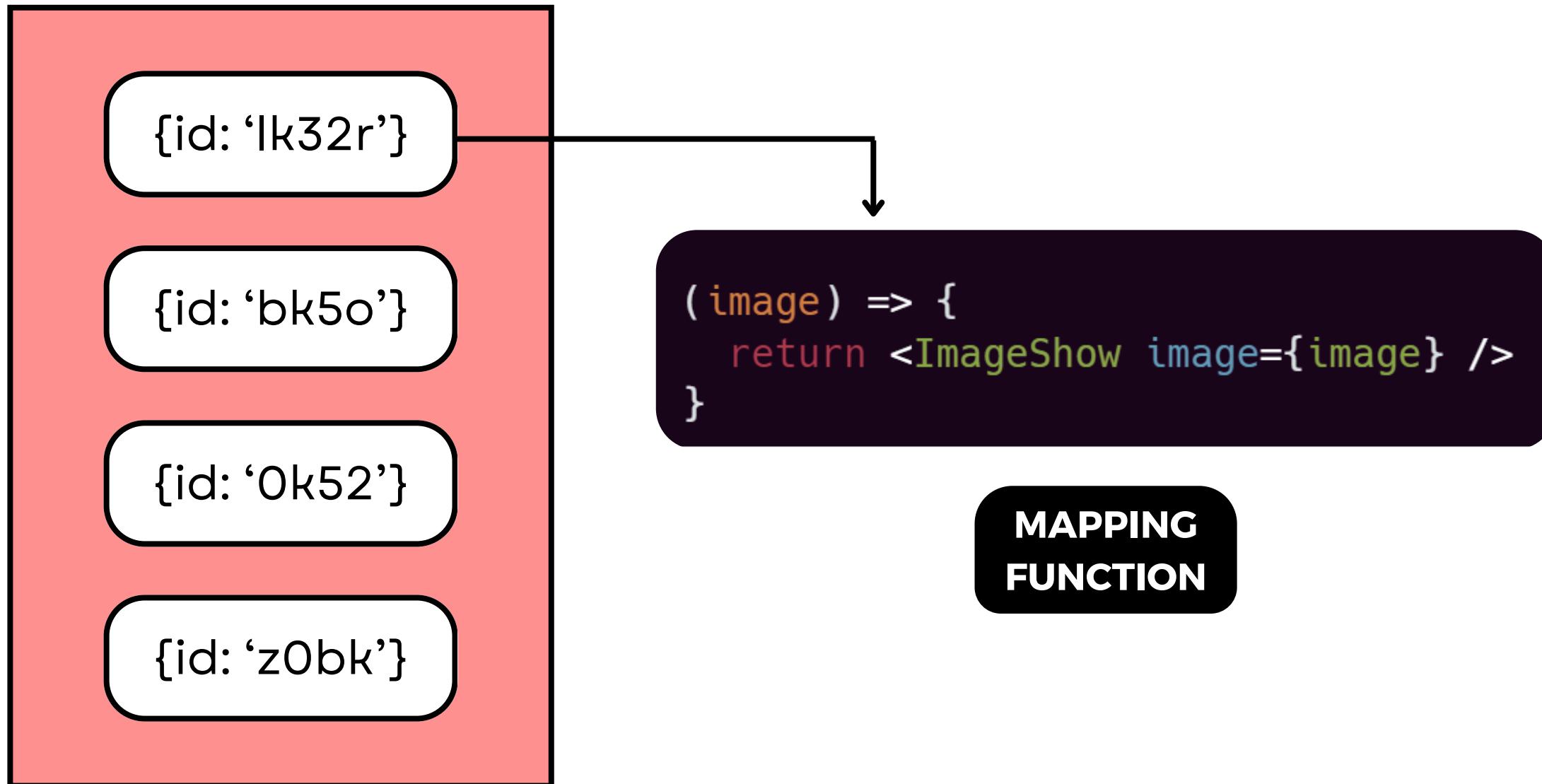
## renderedImages

The diagram shows the result of the mapping process. On the right, a yellow box labeled "renderedImages" contains two identical components, each represented by a rounded rectangle with a black border and white background, containing the rendered image component:

```
<ImageShow image={image} />
```

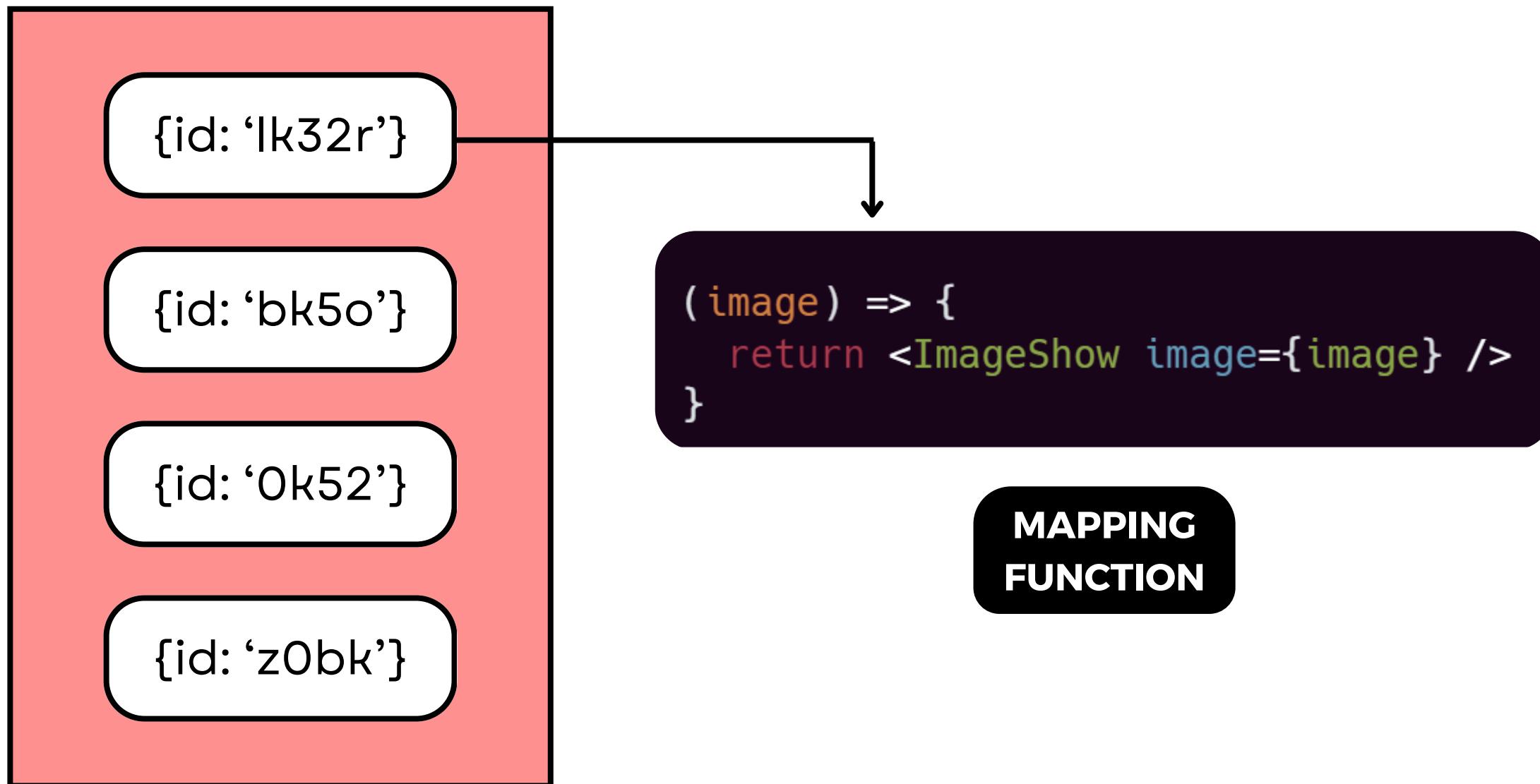
These represent the final output where each original image object has been converted into a rendered `<ImageShow image={image} />` component.

## images

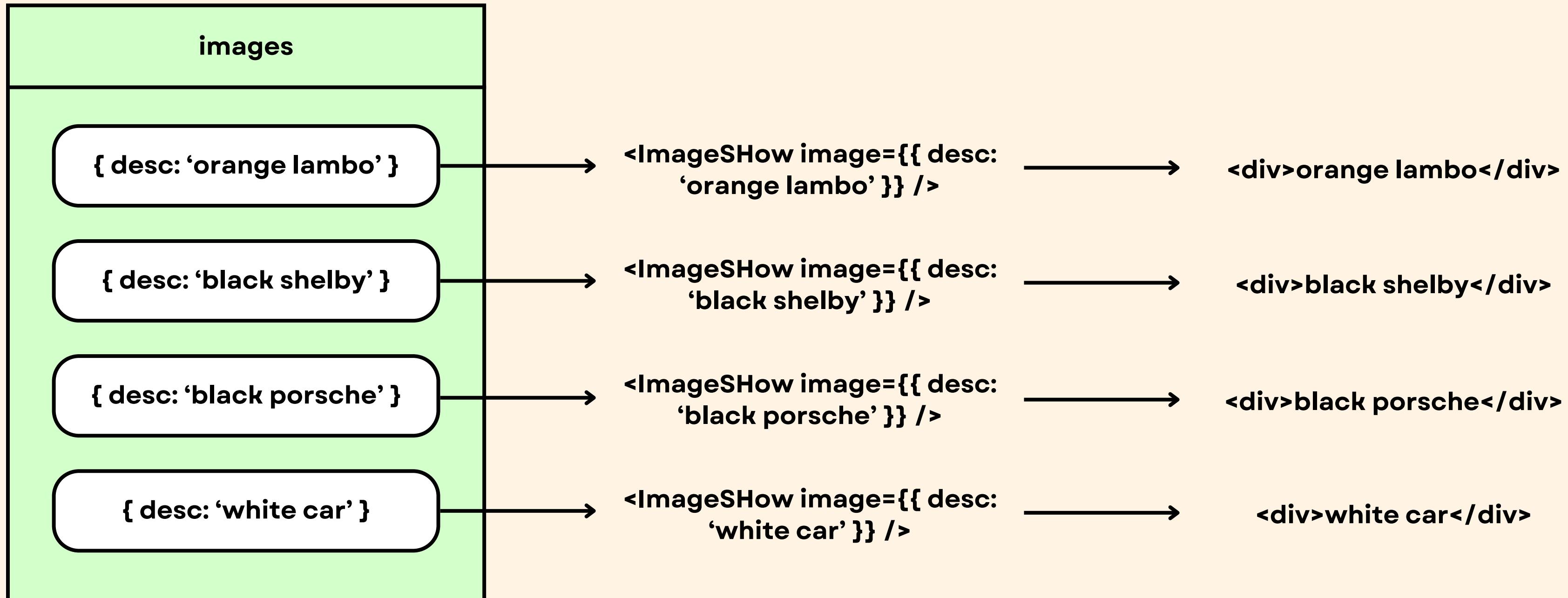


## renderedImages

## images

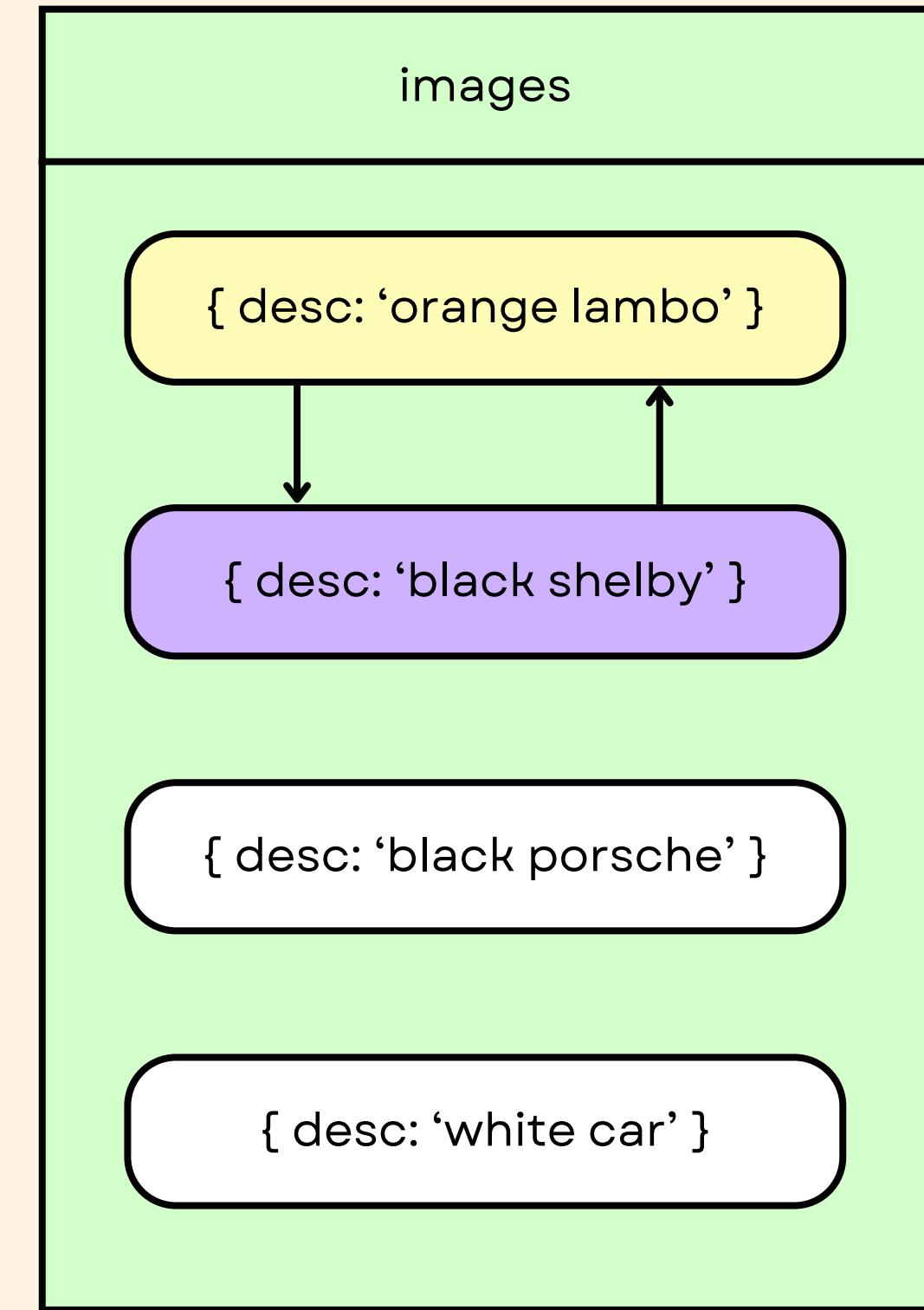


## renderedImages



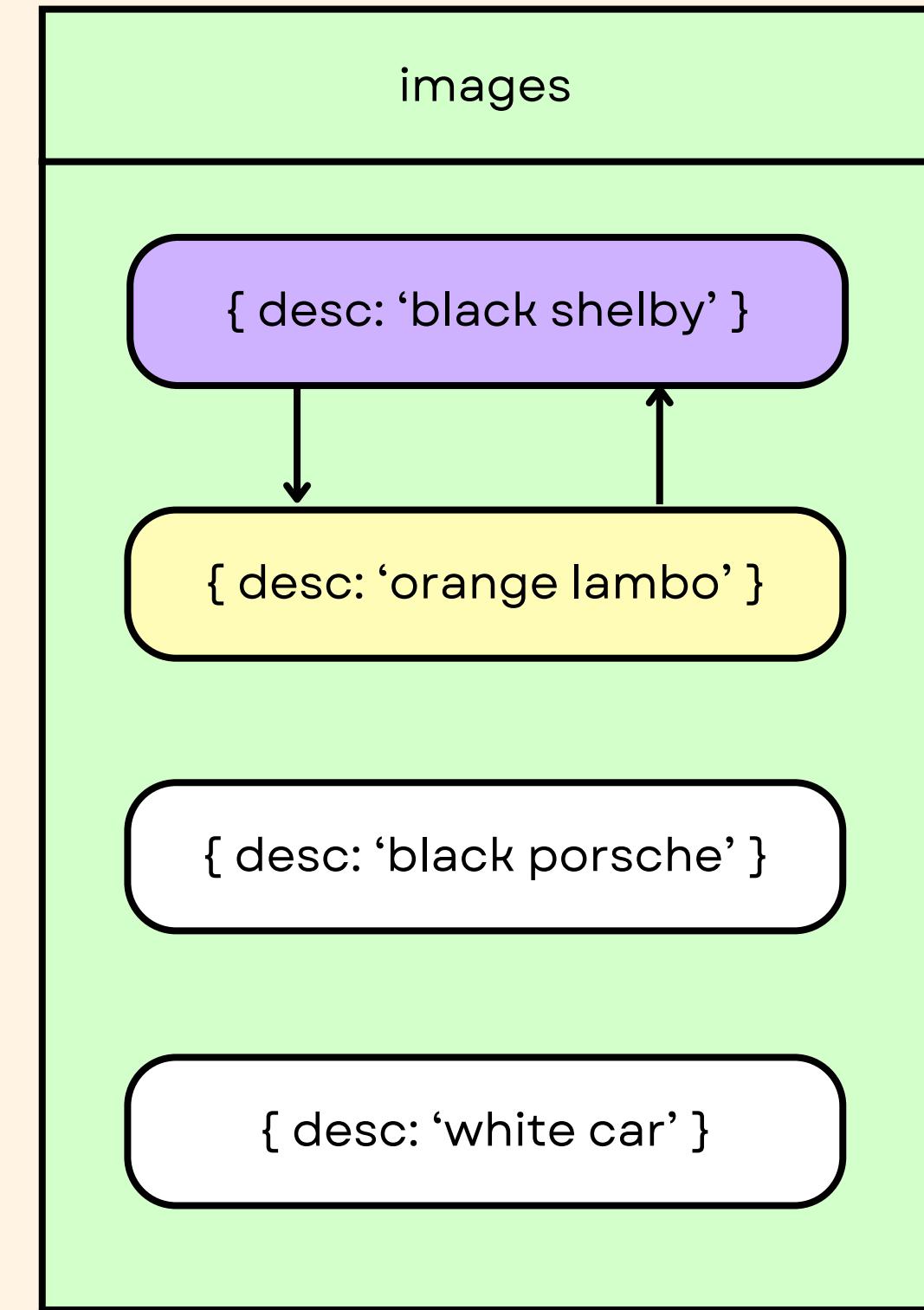
# State มีการเปลี่ยนแปลง!

## เราไม่สามารถอัปเดต HTML!



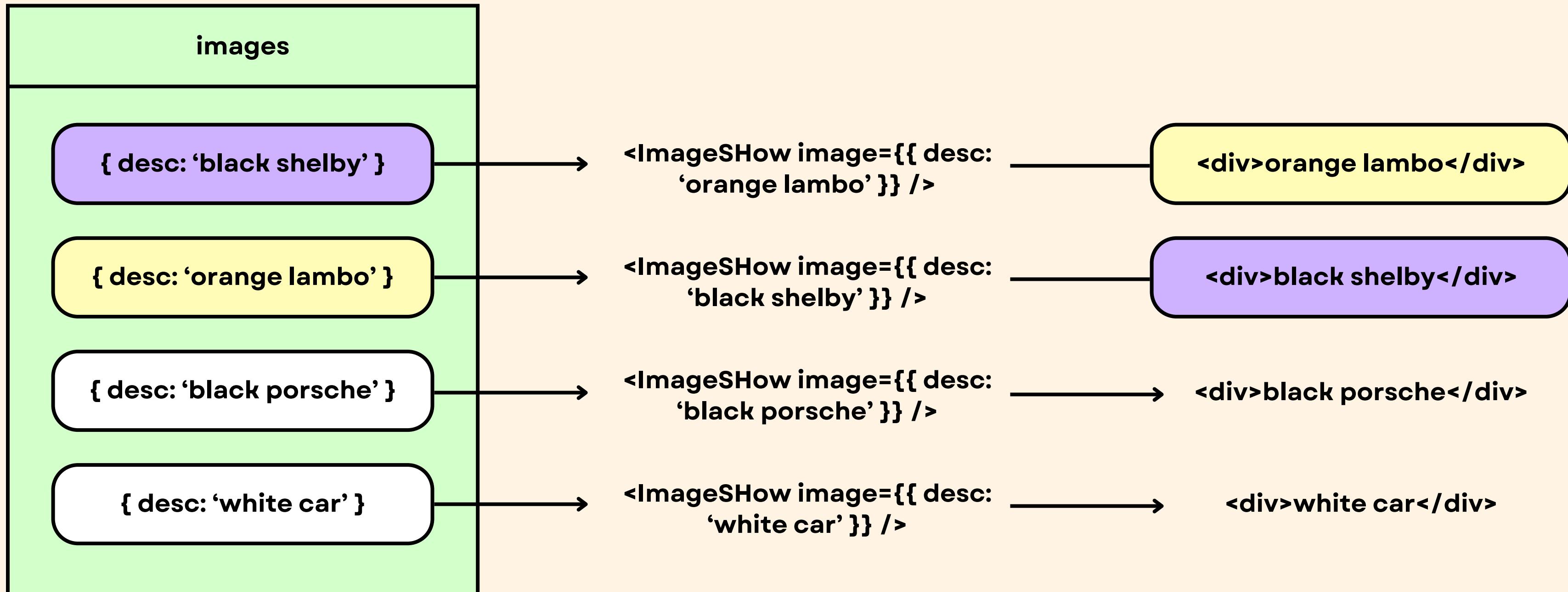
# State มีการเปลี่ยนแปลง!

## เราไม่สามารถอัปเดต HTML!



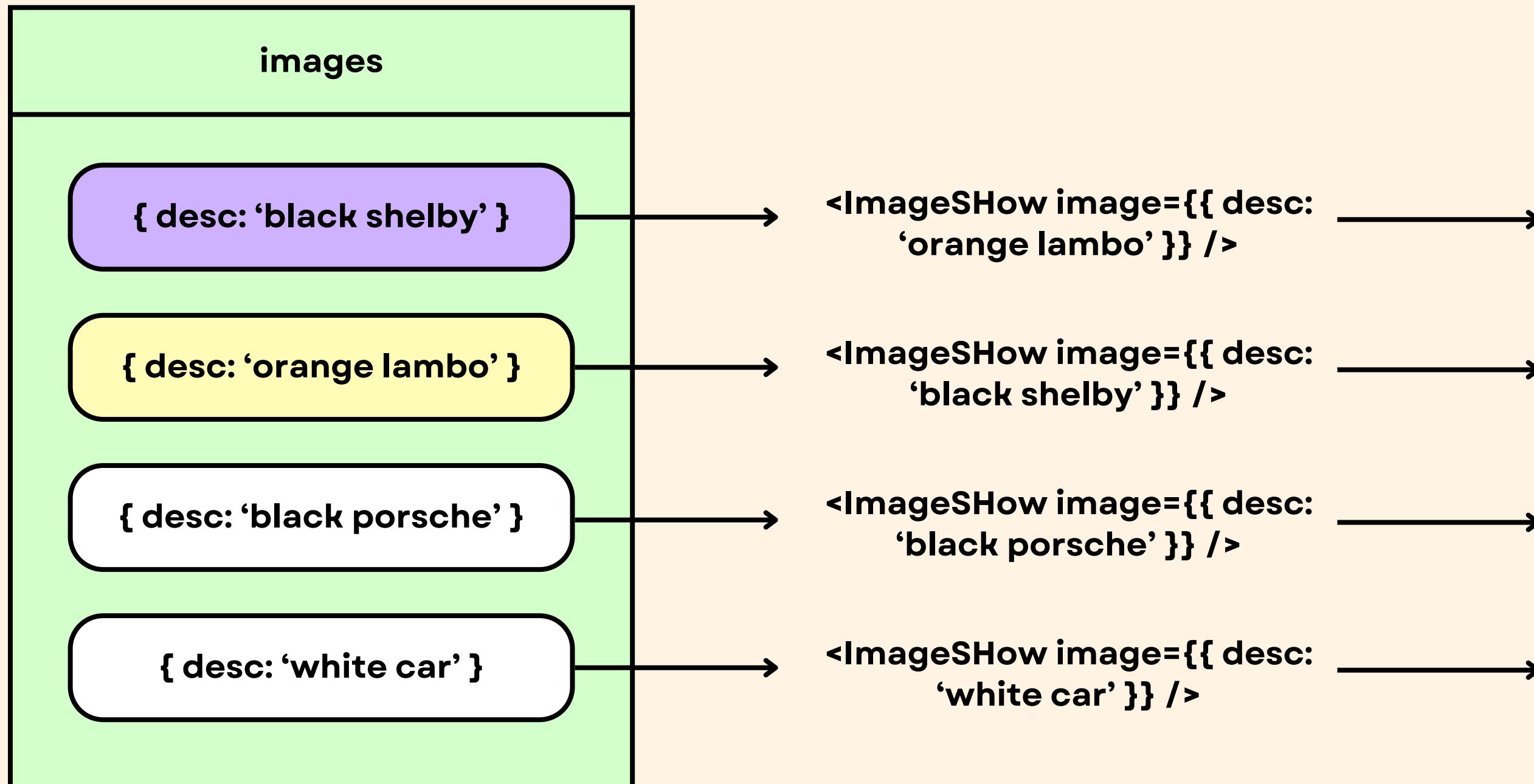
# ตัวเลือก A สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 1: 用 HTML ที่มีอยู่ทั้งหมดที่เกี่ยวข้องกับรายการนี้



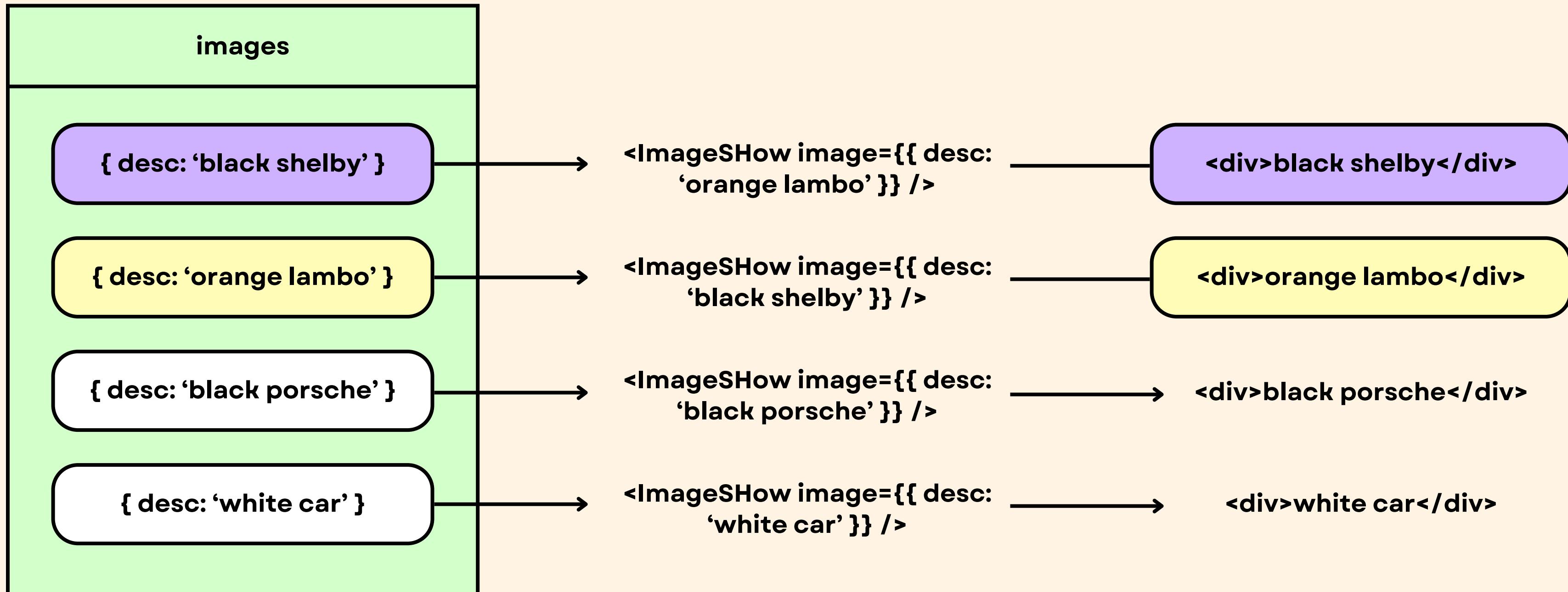
# ตัวเลือก A สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 1: 用 HTML ที่มีอยู่ทั้งหมดที่เกี่ยวข้องกับรายการนี้



# ตัวเลือก A สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 1: au HTML ที่มีอยู่ก็งมงดที่เกี่ยวข้องกับรายการนี้



# ตัวเลือก A สำหรับการใช้งานการอัปเดต

- ขั้นตอนที่ 1: ลบ HTML ที่มิอยู่ทั้งหมดที่เกี่ยวข้องกับ List นี้
- ขั้นตอนที่ 2: สร้างองค์ประกอบ HTML ทั้งหมดใหม่

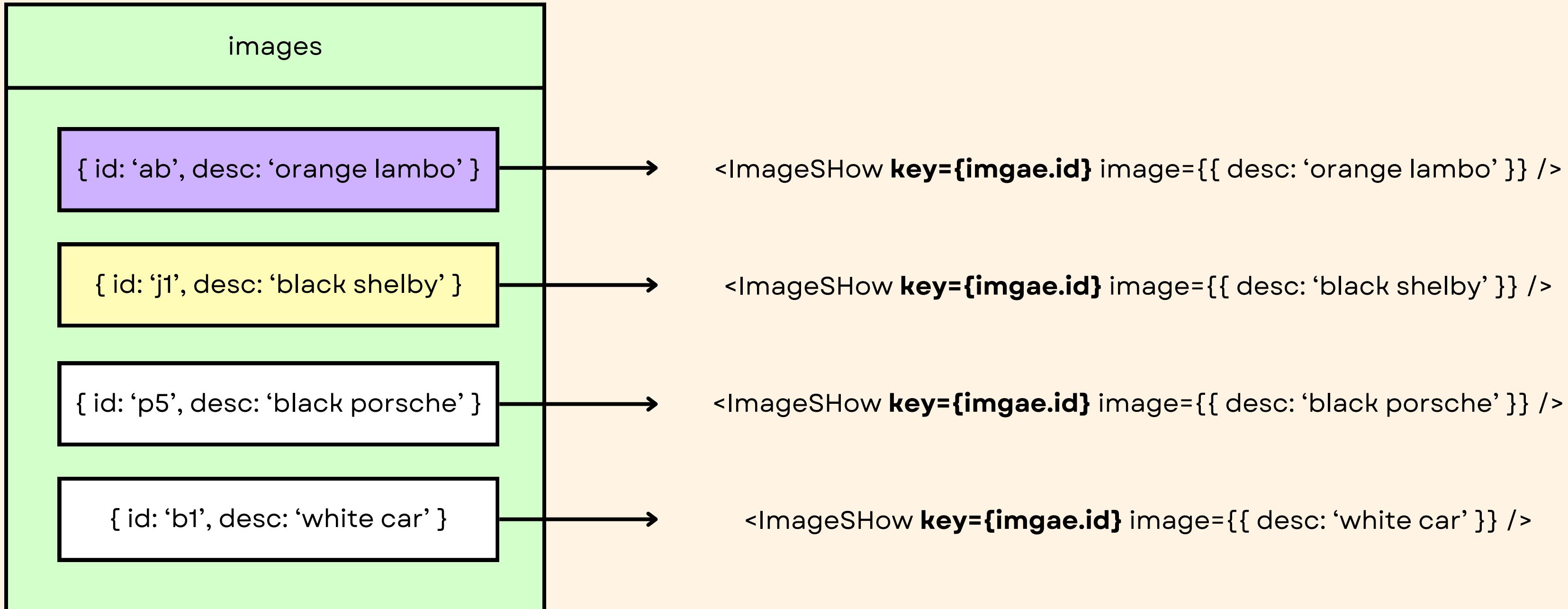
ใช้ได้ผล แต่ใช้ทรัพยากรบุคคลมาก!

ถ้า List ของเรามีสมาชิก 1000 ตัว เราต้องสร้างสมาชิกใหม่ทั้ง 1000 ตัวเพื่อที่จะสลับที่เพียง 2 ตัว...



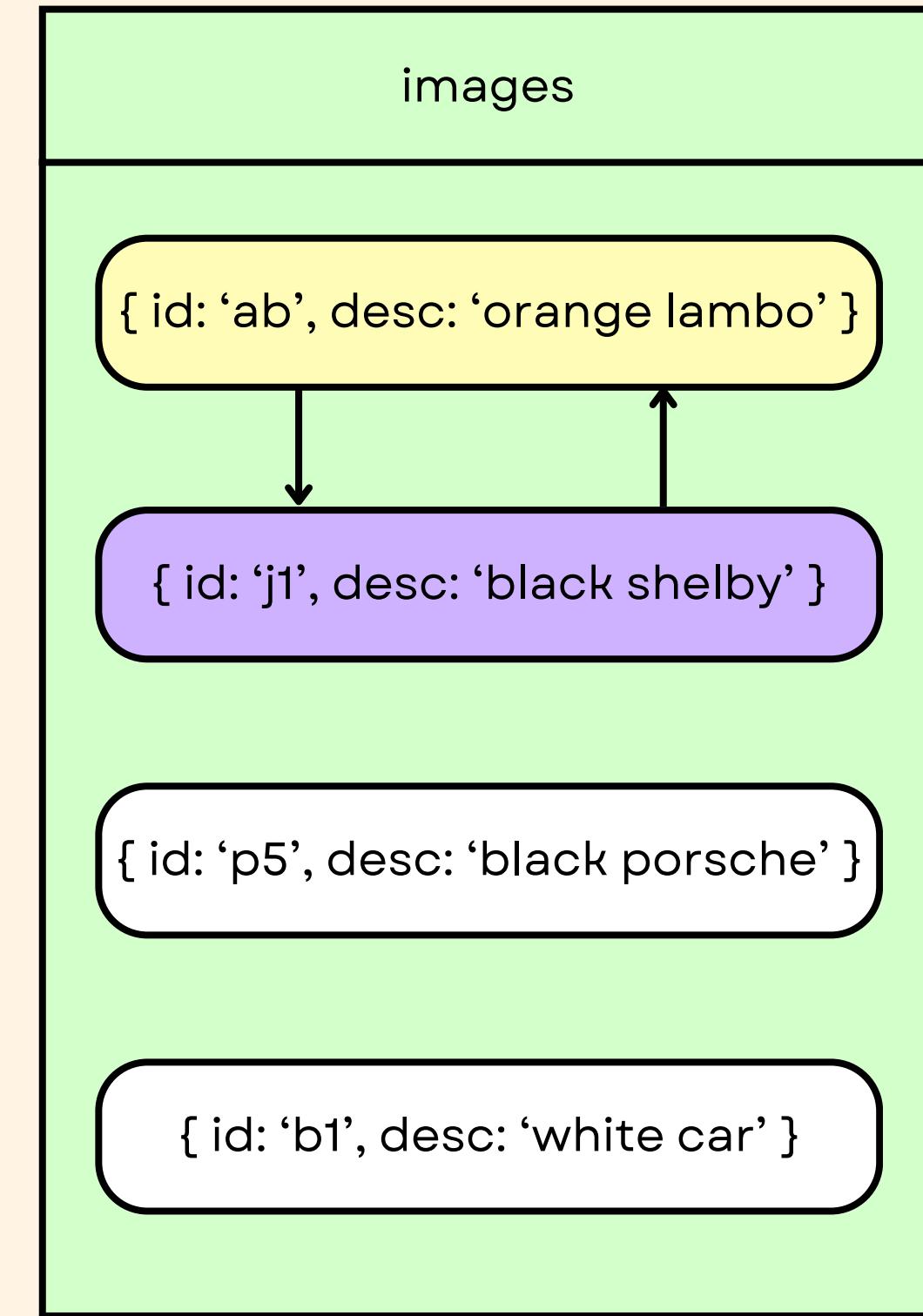
# ตัวเลือก B สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 0: ใส่ "Key" ให้กับสมาชิกแต่ละตัว ตอน Mapping



# State มีการเปลี่ยนแปลง!

## เราไม่สามารถอัปเดต HTML!



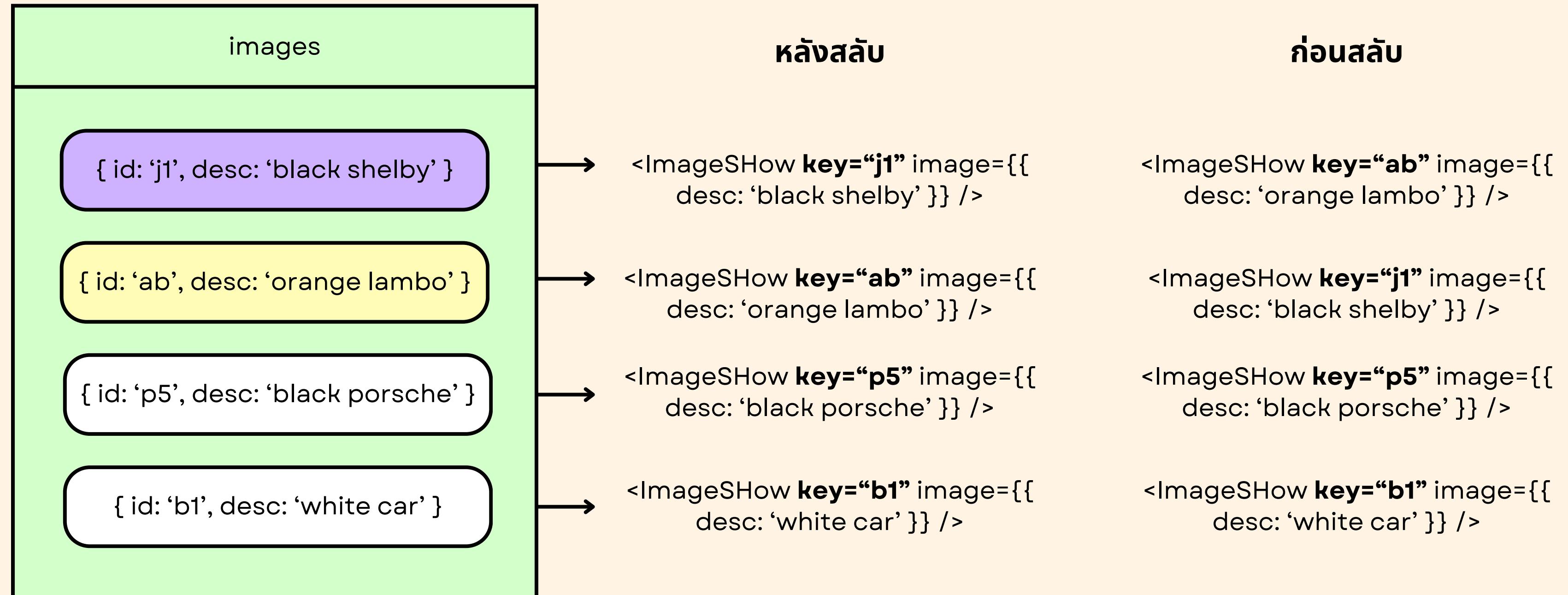
# State มีการเปลี่ยนแปลง!

## เราไม่จำเป็นต้องอัปเดต HTML!



# ตัวเลือก B สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 1: หลังจากเรนเดอร์ใหม่ เปรียบเทียบคีย์ขององแต่ง ImageShow กับคีย์จากการเรนเดอร์ครั้งก่อน (ก่อนสลับ)



# ตัวเลือก B สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 2: ไปแทรกใน DOM เท่าที่จำเป็น (แค่สองอันในกรณีนี้ ไม่ต้องทำใหม่ทั้งหมด)

หลังสลับ

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

ก่อนสลับ

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

HTML ที่มีอยู่  
จากการเรนเดอร์ครั้งแรก

<div>orange lambo</div>

<div>black shelby</div>

<div>white car</div>

<div>black porsche</div>

# ตัวเลือก B สำหรับการใช้งานการอัปเดต

ขั้นตอนที่ 2: ไปแทรกใน DOM เท่าที่จำเป็น (แค่สองอันในกรณีนี้ ไม่ต้องทำใหม่ทั้งหมด)

หลังสลับ

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

ก่อนสลับ

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

HTML ที่มีอยู่  
จากการเรนเดอร์ครั้งแรก

<div>orange lambo</div>



<div>black shelby</div>

<div>white car</div>

<div>black porsche</div>

# ตัวเลือก B สำหรับการใช้งานการอัปเดต

## ขั้นตอนที่ 2: ใช้ชุดการเปลี่ยนแปลงขึ้นต่ำกับองค์ประกอบ DOM ที่มีอยู่

หลังสลับ

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

ก่อนสลับ

```
<ImageShow key="ab" image={{  
    desc: 'orange lambo' }} />
```

```
<ImageShow key="j1" image={{  
    desc: 'black shelby' }} />
```

```
<ImageShow key="p5" image={{  
    desc: 'black porsche' }} />
```

```
<ImageShow key="b1" image={{  
    desc: 'white car' }} />
```

HTML ที่มีอยู่  
จากการเรนเดอร์ครั้งแรก

<div>black shelby</div>

<div>orange lambo</div>

<div>white car</div>

<div>black porsche</div>



# ตัวเลือก B สำหรับการใช้การ Update

## เราทำแค่นี้

- ใส่ "Key" ให้กับสมาชิกแต่ละตัว ตอน Mapping

## ที่เหลือ React ทำให้

- หลังจากเรนเดอร์ใหม่ เปรียบเทียบคีย์ของแท็ลส ImageShow กับคีย์จากการเรนเดอร์ครั้งก่อน (ก่อนสลับ)
- ไปแก้ไข DOM เท่าที่จำเป็น



# ตัวเลือก B สำหรับการใช้การ Update

## เราทำแค่นี้

- ใช้ "Key" ให้กับสมาชิกแต่ละตัว ตอน Mapping

# ข้อกำหนดของ Keys

- ใช้เมื่อมี Array ของ Elements (ทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยัง JSX ที่อยู่ด้านบนสุด
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับ Array นี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ (ไม่มีการเปลี่ยนแปลง)



ตัวนี้เป็น Element ตัว  
บุสุด

```
function App() {  
  const images = [{ id: 'a1' }, { id: 'b2' }];  
  
  const renderedImages = imgaes.map((image) => {  
    return <ImageShow key={image.id} />;  
  });  
  
  return (  
    <div>  
      {renderedImages}  
    </div>  
  );  
}
```

มีการเปลี่ยนโค๊ดใหม่ (Refactor)  
ซึ่งตอนนี้ key ยังอยู่ใน ImageShow  
(แต่ตอนนี้ ImageShow ไม่ได้เป็น Element ตัว  
หนึ่งแล้ว)

```
function App() {  
  const images = [{ id: 'a1' }, { id: 'b2' }];  
  
  const renderedImages = images.map((image) => {  
    return (  
      <div>  
        <ImageShow key={image.id} />  
      </div>  
    );  
  });  
  
  return (  
    <div>  
      {renderedImages}  
    </div>  
  );  
}
```

## ຢ້າຍ key ມາ Element ທີ່ອຸ່ນບຸນຊຸດ

```
function App() {  
  const images = [t id: 'a1', id: 'b2' ];  
  
  const renderedImages = images.map((image) => {  
    return (  
      <div key={image.id}>  
        <ImageShow />  
      </div>  
    )  
  })  
  
  return (  
    <div>  
      {renderedImages}  
    </div>  
  )  
}
```

# ข้อกำหนดของ Keys

- ใช้เมื่อมี Array ของ Elements (ทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยัง JSX ที่อยู่ด้านบนสุด
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับ Array นี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ (ไม่มีการเปลี่ยนแปลง)



สอง Array กี่มี  
id เมื่อันกัน

ใช้ id เป็น key  
ก็สอง Array

เวลาเรนเดอร์ Array ยังแยก  
กันอยู่ จึงไม่เป็นปัญหา OK!

```
function App() {  
  const images = [{ id: 'a1' }, { id: 'b2' }];  
  const videos = [{ id: 'a1' }, { id: 'b2' }];  
  
  const renderedImages = images.map((image) => {  
    return <div key={image.id} />;  
  });  
  
  const renderedVideos = videos.map((video) => {  
    return <div key={video.id} />;  
  });  
  
  return (  
    <div>  
      {renderedImages}  
      {renderedVideos}  
    </div>  
  );  
}
```

สอง Array ที่มี  
id เมื่อันกัน

ทั้งสองรายการ  
ใช้ id เป็น Key

มีการรวม array ทั้งสองอัน  
เข้าด้วยกันเป็นอันเดียว ทำให้  
มี key ซ้ำกันใน Array นี้

```
function App() {  
  const images = [{ id: 'a1' }, { id: 'b2' }];  
  const videos = [{ id: 'a1' }, { id: 'b2' }];  
  
  const renderedImages = images.map((image) => {  
    return <div key={image.id} />;  
  });  
  
  const renderedVideos = videos.map((video) => {  
    return <div key={video.id} />;  
  });  
  
  return (  
    <div>  
      {renderedImages.concat(renderedVideos)}  
    </div>  
  );  
}
```

# ข้อกำหนดของ Keys

- ใช้เมื่อมี Array ของ Elements (ทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยัง JSX ที่อยู่ด้านบนสุด
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับ Array นี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ (ไม่มีการเปลี่ยนแปลง)



```
function App() {
  const images = [{ id: 'a1' }, { id: 'b2' }];

  const renderedImages = images.map((image) => {
    return (
      <div key={Math.random()}>
        <ImageShow />
      </div>
    );
  });

  return (
    <div>
      {renderedImages}
    </div>
  );
}
```

Key ที่สร้างขึ้นแบบสุ่ม รายการจะถูก  
สร้างใหม่ทุกครั้งเมื่อมีการเรนเดอร์

# ความต้องการสำหรับ Keys

- ใช้เมื่อเรามีรายการขององค์ประกอบ (ซึ่งหมายถึงทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยังองค์ประกอบ JSX ก่อนอยู่ด้านบนสุดของการ
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับรายการนี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ใหม่



images

{id: 'ab', ...other...}

{id: 'b2', ...other...}

{id: 'c7', ...other...}

{id: '5e', ...other...}

ส่วนใหญ่ข้อมูลที่เป็น Array

จะถูกดึงออกมาจาก Database

messages

{id: 'ab', ...other...}

{id: 'b2', ...other...}

{id: 'c7', ...other...}

{id: '5e', ...other...}

videos

{id: 'ab', ...other...}

{id: 'b2', ...other...}

{id: 'c7', ...other...}

{id: '5e', ...other...}

ข้อมูลเหล่านี้จะมี id ที่ unique อยู่แล้ว ซึ่งเราเอาตัวนี้มาใช้ได้เลย

ในกรณีที่ไม่มี  
ID มาให้ใน  
**Array**

การใช้ *index* เป็น *key*  
จะทำให้เกิดบัคภายใน  
ที่สุด

## ใช้ index ของ Array



```
function App() {  
  const values = [];  
  
  for (let i = 0; i < 5; i++) {  
    values.push(<div key={i}></div>);  
  }  
  
  return (  
    <div>  
      {values}  
    </div>  
  );  
}
```

## สร้าง unique ID ขึ้นมาเอง



เดี๋ยวเมตัวอย่างให้ดูในบท  
ถัด ๆ ไป



# ความต้องการสำหรับ Keys

- ใช้เมื่อเรามีรายการขององค์ประกอบ (ซึ่งหมายถึงทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยังองค์ประกอบ JSX ก่อนอยู่ด้านบนสุดของการ
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับรายการนี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ใหม่



# การจัดการข้อความ Inputs

1. สร้าง State ขึ้นมาใหม่

2. สร้าง event handler เพื่อสำหรับดัก Event  
'onChange'

3. ในการดึงค่าจาก input ใช้ Event ที่ได้จาก  
'onChange'

4. นำค่านั้นมาอัปเดตใน State

5. ส่ง State ไปยัง input ใน property ที่ชื่อว่า  
'value'

# ทำไม???

S

เราสามารถจัดการ  
text ที่อยู่ใน input  
ได้โดยใช้ state

N

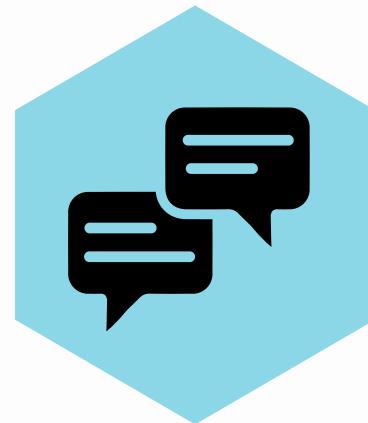
ถ้าต้องการอ่านค่า  
ของ input ดูที่  
state ('term')

N

ถ้าต้องการอัปเดตค่า  
ของ input ให้ใช้  
'setTerm('Hello')'

I

การเขียน Inputs  
แบบนี้เรียกว่า  
"controlled  
inputs"



สื่อสารจาก Child ไปยัง  
Parent



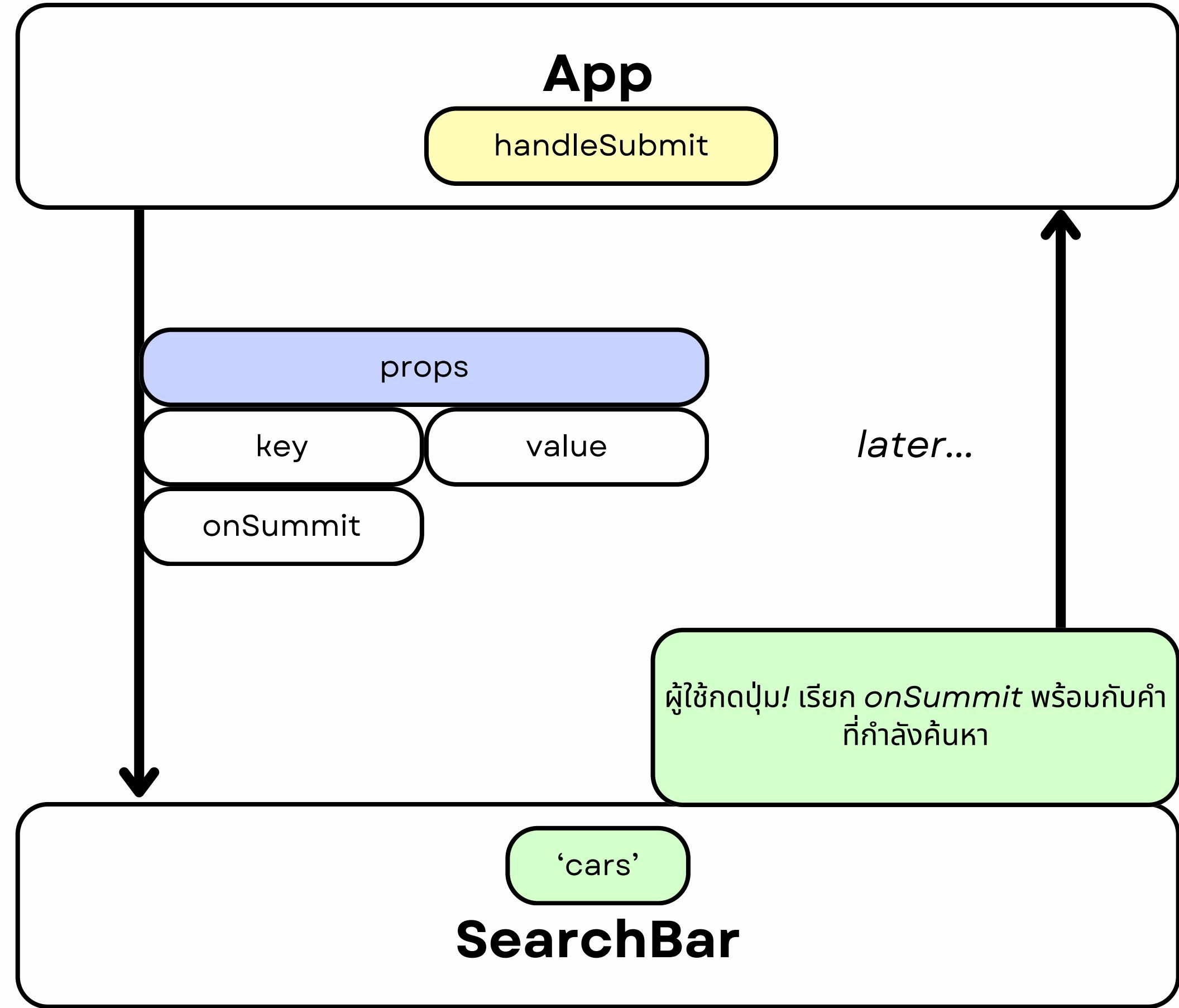
เหมือนกับ event  
ธรรมดาเลย

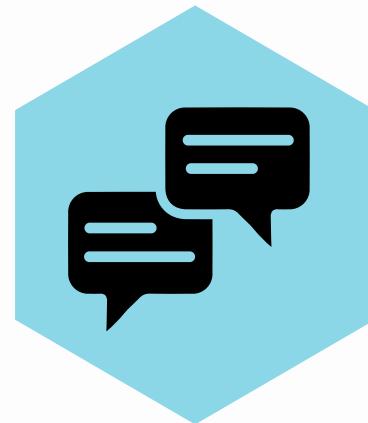


ส่ง Event Handler  
ลงไปที่ Child



เรียกใช้ Event Handler  
นี้เมื่อต้องการส่งค่าขึ้นไป





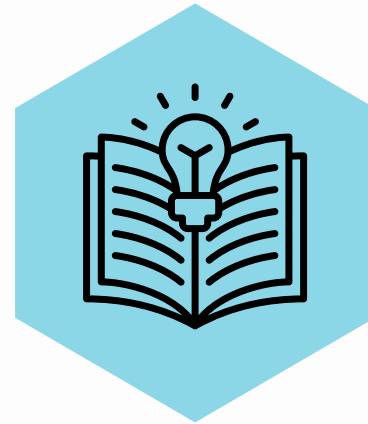
สื่อสารจาก Child ไปยัง  
Parent



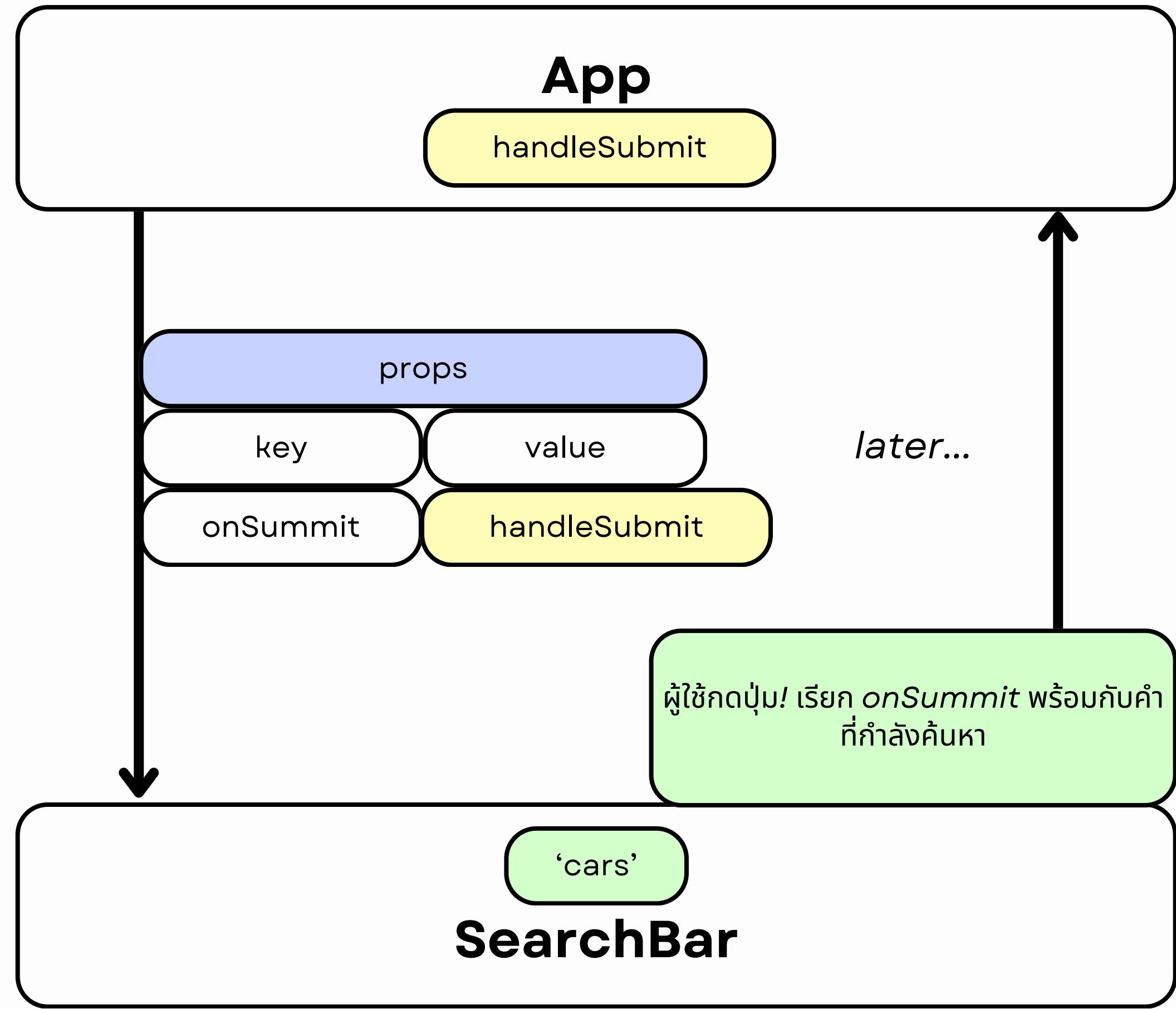
เหมือนกับ event  
ธรรมดาเลย

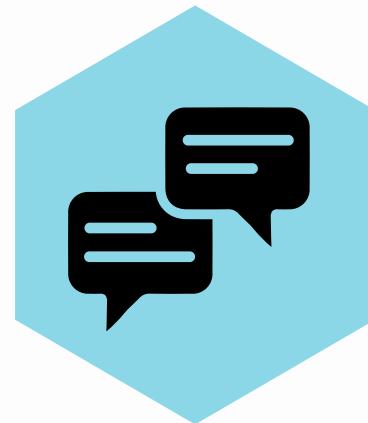


ส่ง Event Handler  
ลงไปที่ Child



เรียกใช้ Event Handler  
นี้เมื่อต้องการส่งค่าขึ้นไป





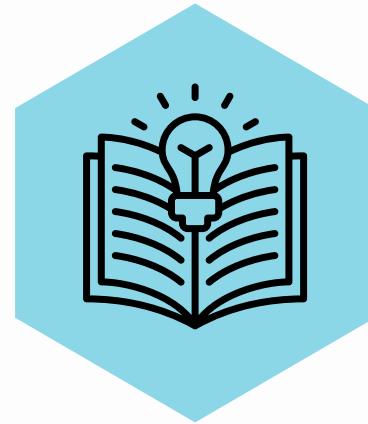
สื่อสารจาก Child ไปยัง  
Parent



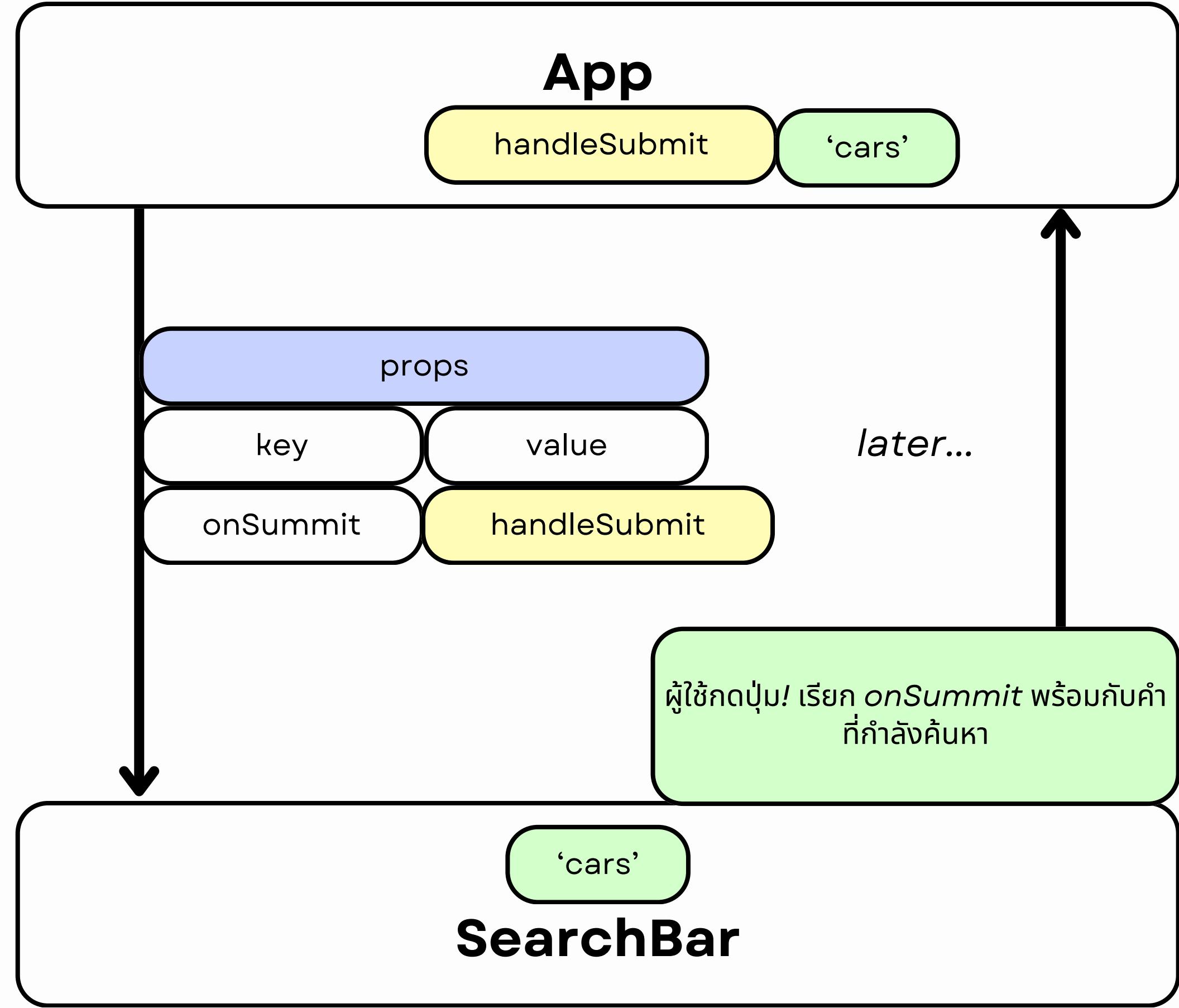
เหมือนกับ event  
ธรรมดาเลย



ส่ง Event Handler  
ลงไปที่ Child



เรียกใช้ Event Handler  
นี้เมื่อต้องการส่งค่าขึ้นไป



# ข้อกำหนดของ Keys

- ใช้เมื่อมี Array ของ Elements (ทุกครั้งที่เราใช้งาน 'map')
- เพิ่ม keys ไปยัง JSX ที่อยู่ด้านบนสุด
- จะต้องเป็นสตริงหรือตัวเลข
- ควรจะมีค่าที่ไม่ซ้ำกับสำหรับ Array นี้
- ควรมีความสม่ำเสมอตลอดการเรนเดอร์ (ไม่มีการเปลี่ยนแปลง)

