


Bsidesoft co.

since 2004


ASYNC AWAIT

Sync

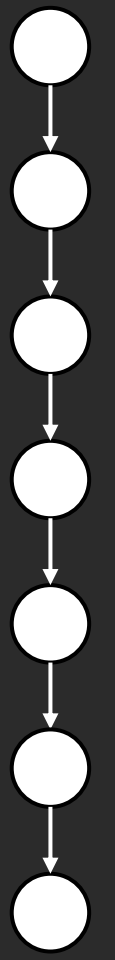
Sync flow control

Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync flow control

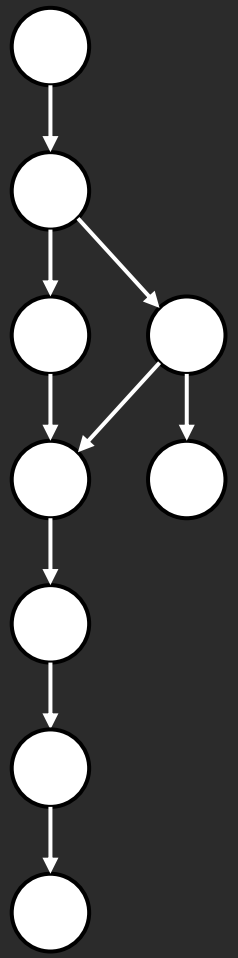
Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨



Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync Flow Control: Goto를 통해 명령의 위치를 이동함

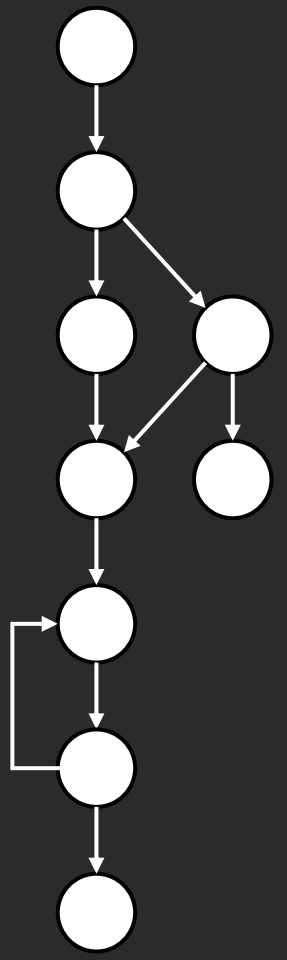


Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync Flow Control: Goto를 통해 명령의 위치를 이동함

Sub Flow: 함수 등을 통해 별도의 명령셋을 여러번 실행함

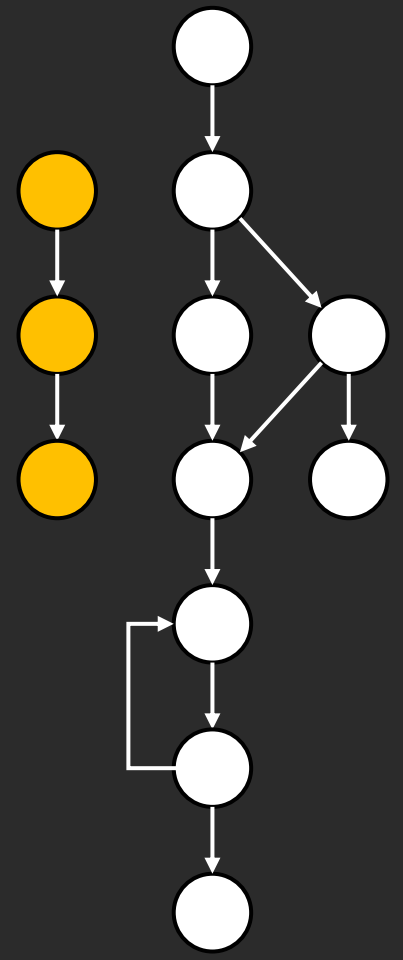


Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync Flow Control: Goto를 통해 명령의 위치를 이동함

Sub Flow: 함수 등을 통해 별도의 명령셋을 여러번 실행함

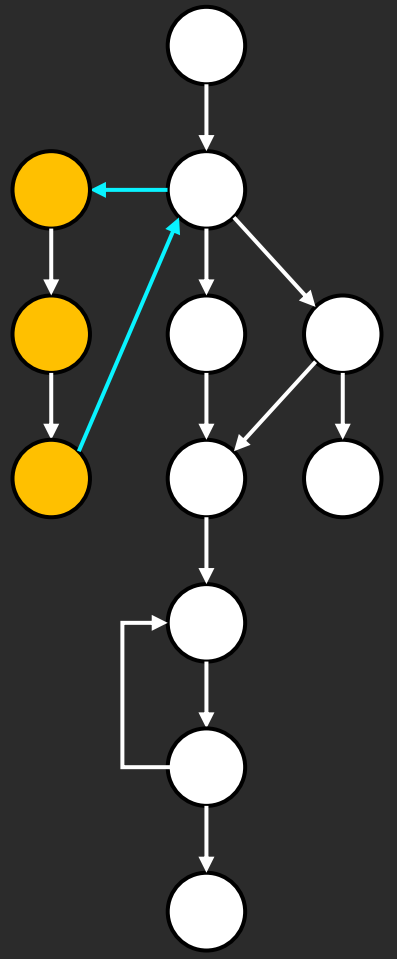


Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync Flow Control: Goto를 통해 명령의 위치를 이동함

Sub Flow: 함수 등을 통해 별도의 명령셋을 여러번 실행함

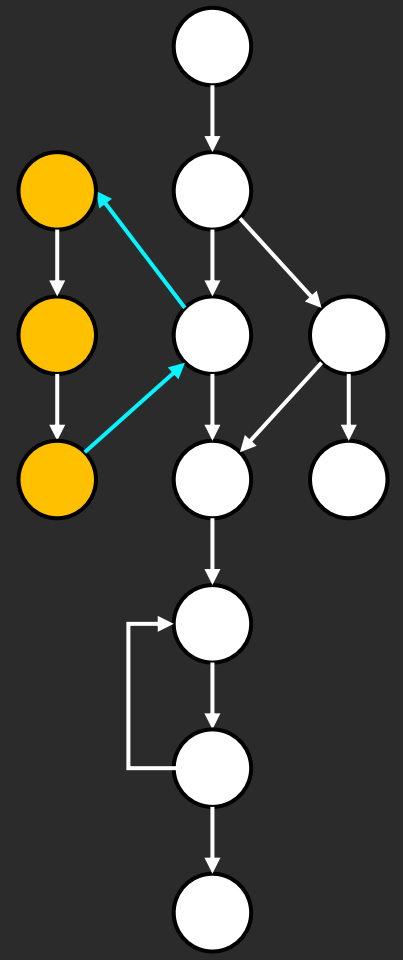


Sync flow control

Sync Flow: 메모리에 적재된 명령이 순차적으로 실행됨

Sync Flow Control: Goto를 통해 명령의 위치를 이동함

Sub Flow: 함수 등을 통해 별도의 명령셋을 여러번 실행함



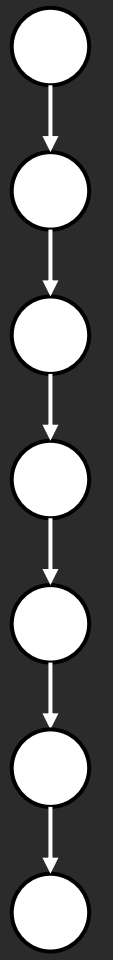
Blocking

Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

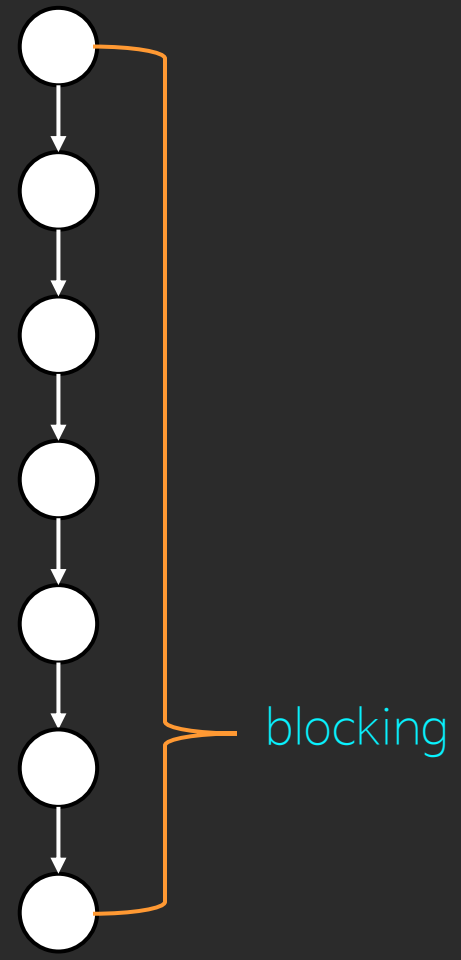
Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상



Blocking

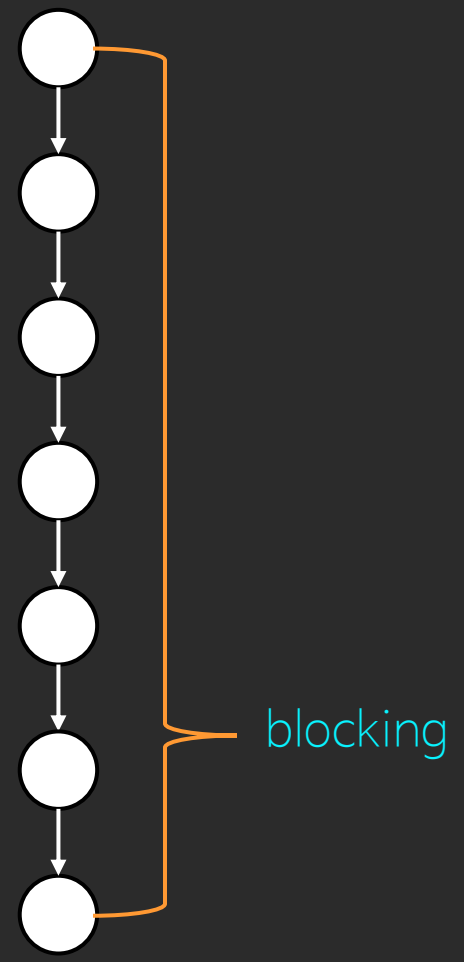
Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상



Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

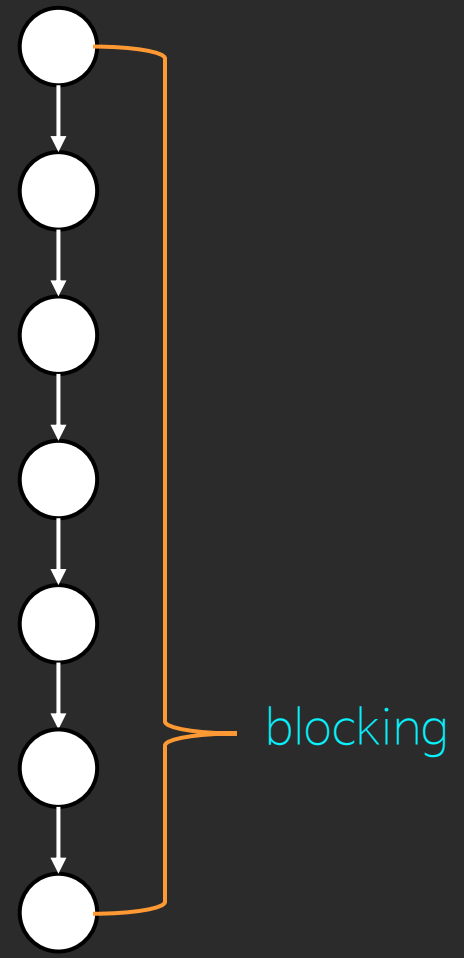


Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

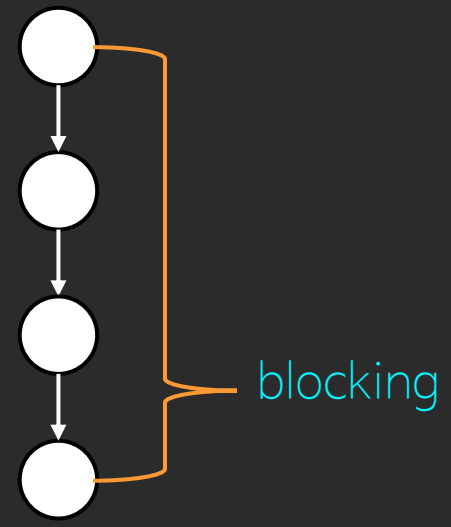


Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기



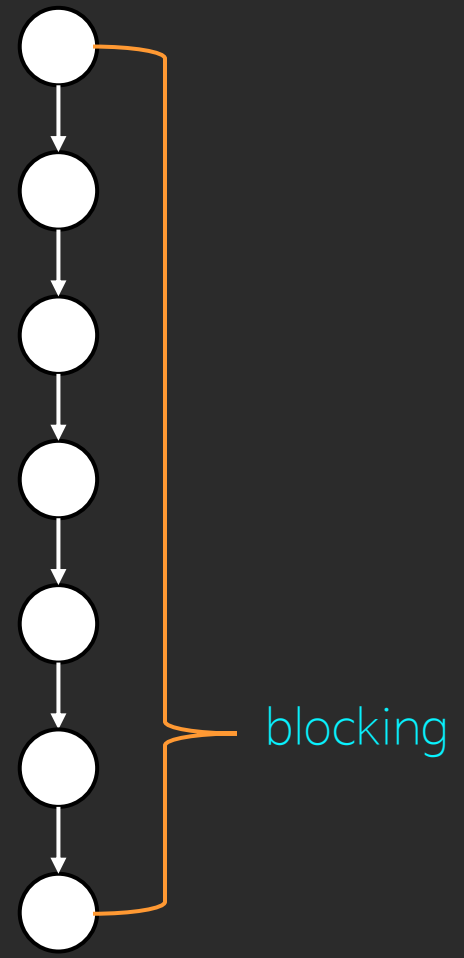
Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기



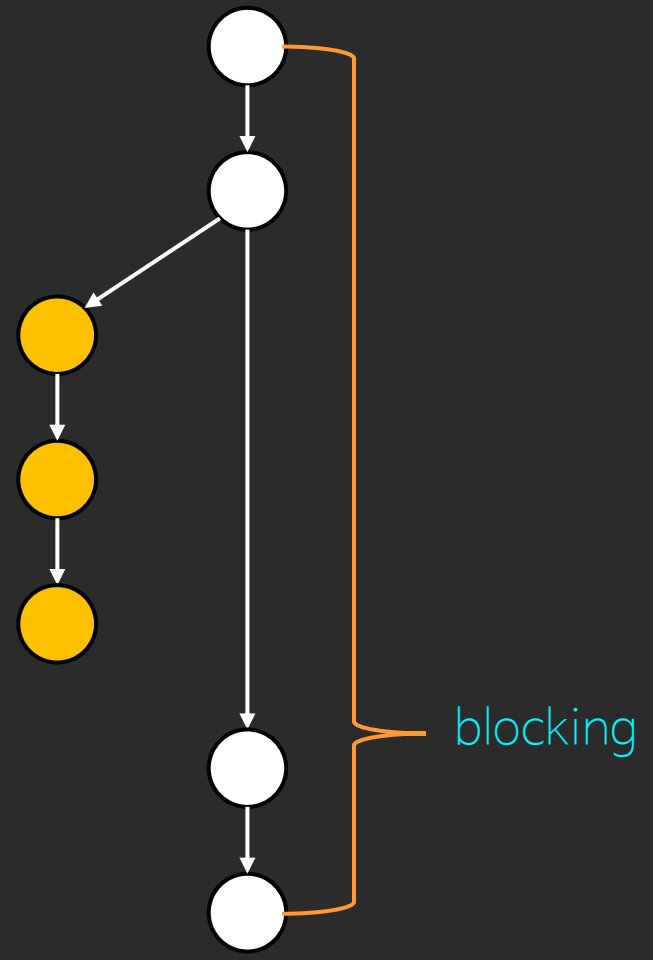
Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기



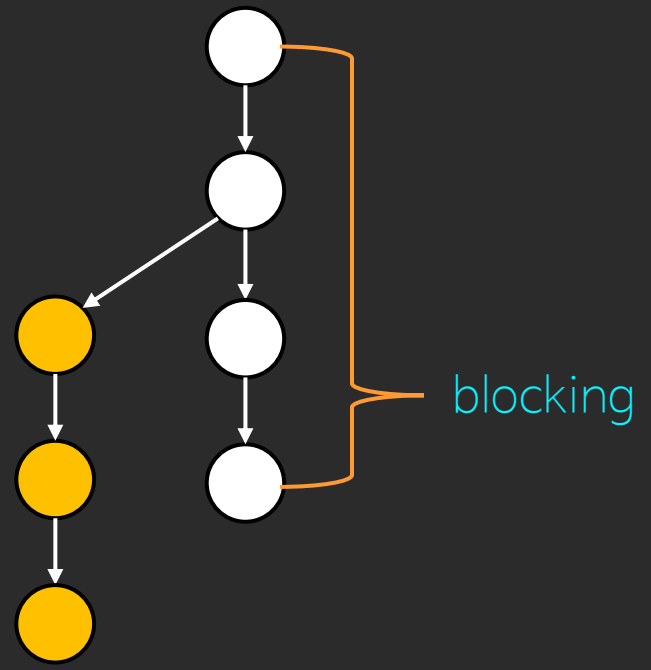
Blocking

Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기



Blocking

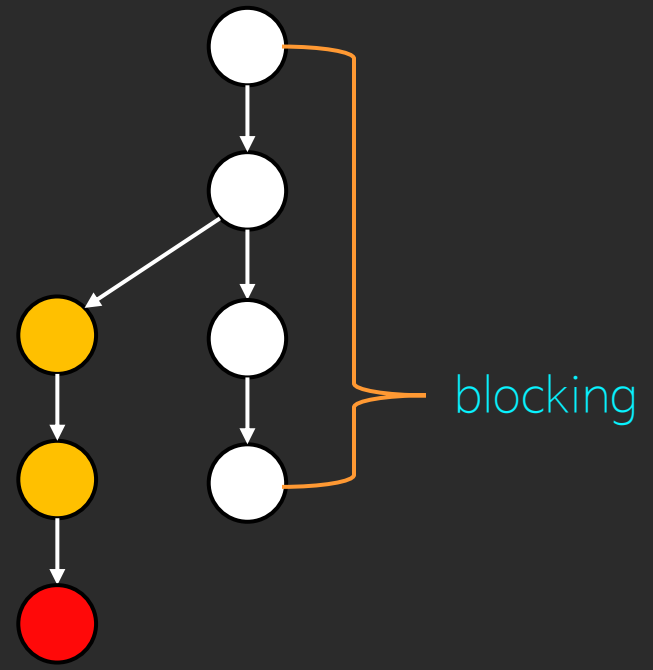
Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기

다른 쓰레드의 작업이 완료되면 원래 쓰레드에 보고해야함



Blocking

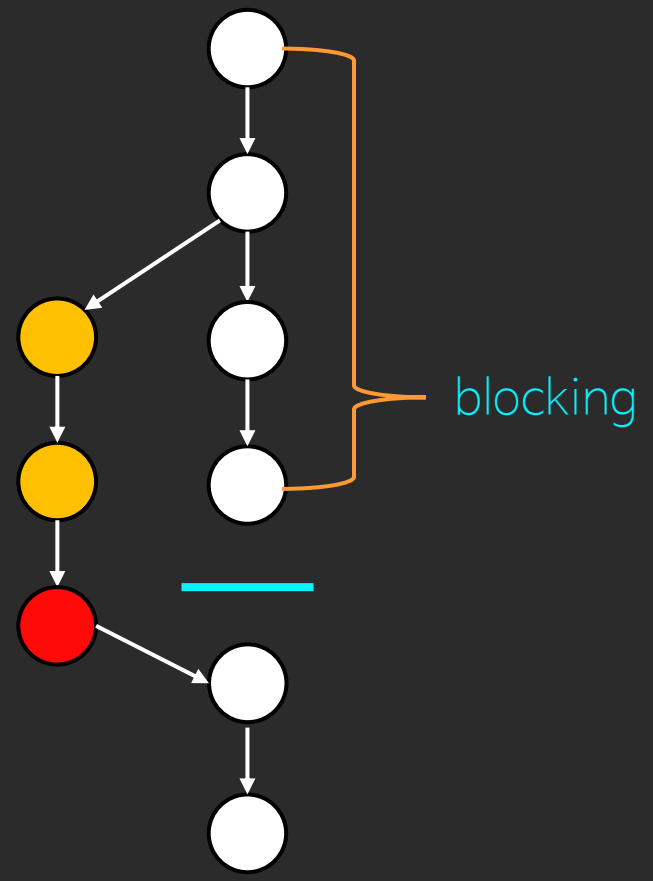
Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기

- 다른 쓰레드의 작업이 완료되면 원래 쓰레드에 보고해야함



Blocking

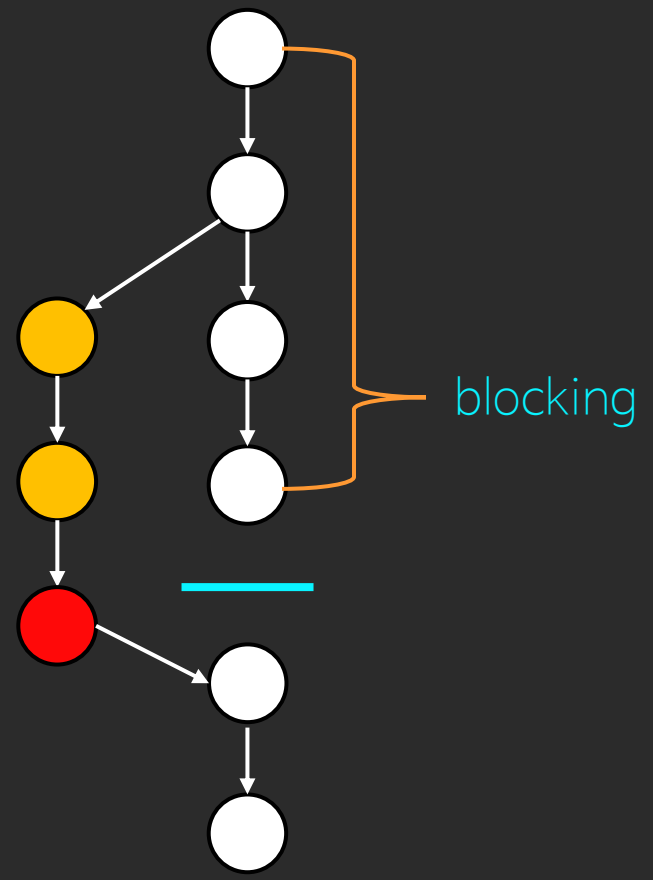
Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기

다른 쓰레드의 작업이 완료되면 원래 쓰레드에 보고해야함



Non Blocking

Blocking

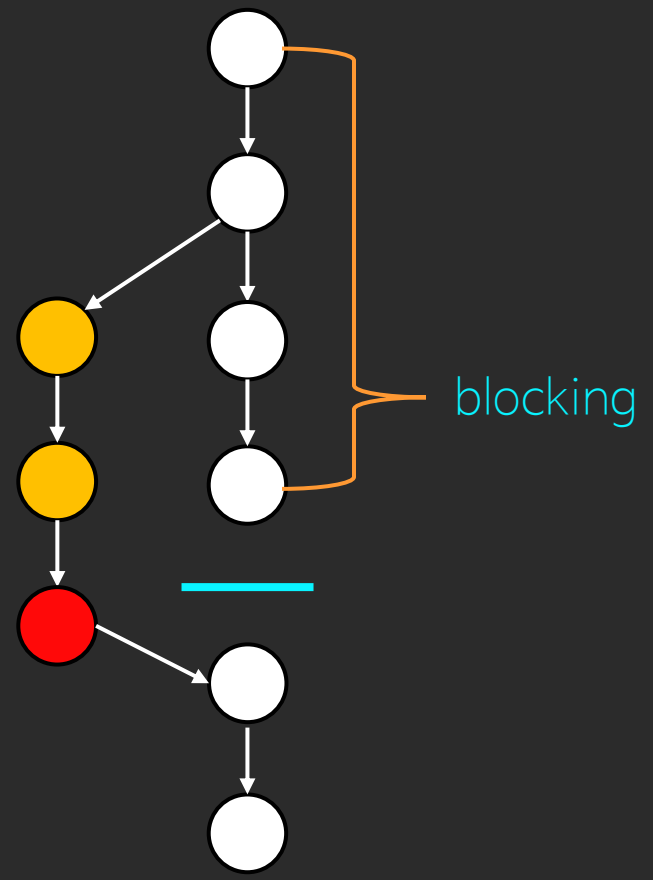
Sync Flow가 실행되는 동안 다른 일을 할 수 없는 현상

Blocking 줄이기

sync flow를 짧게 하기

다른 쓰레드에 syncflow를 떠넘기기

다른 쓰레드의 작업이 완료되면 원래 쓰레드에 보고해야함



Non Blocking

Sync Flow가 납득할 만한 시간 내에 종료되는 것

Async

Sync & Async

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

다른 수단??

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

Promise

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

Promise

callback function

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

Promise

callback function

iterations

Sync & Async

Sync - 서브루틴이 즉시 값을 반환함

Async - 서브루틴이 다른 수단으로 값을 반환함

Promise

callback function

iterations

Async 단점

Async 단점

호출결과가 즉시 반환되지 않으므로 현재의 sync flow가 종료됨

Async 단점

호출결과가 즉시 반환되지 않으므로 현재의 sync flow가 종료됨

그 결과 현재의 어휘공간 내의 상태를 결과시점에 사용할 수 없음

Async 단점

호출결과가 즉시 반환되지 않으므로 현재의 sync flow가 종료됨

그 결과 현재의 어휘공간 내의 상태를 결과시점에 사용할 수 없음

요청 시의 상태를 별도로 결과시점에 전달할 부가장치 필요

Sync의 장점 + Async의 장점

Sync의 장점 + Async의 장점

sync로직으로 async를 사용할 수 있게 함

하지만 sync flow가 어긋나므로 이전 sync flow의 상태를 기억하여 이어줄 장치 필요

Sync의 장점 + Async의 장점

sync로직으로 async를 사용할 수 있게 함

하지만 sync flow가 어긋나므로 이전 sync flow의 상태를 기억하여 이어줄 장치 필요

상태를 기억하고 이어주는 장치 – Continuation

Sync의 장점 + Async의 장점

sync로직으로 async를 사용할 수 있게 함

하지만 sync flow가 어긋나므로 이전 sync flow의 상태를 기억하여 이어줄 장치 필요

상태를 기억하고 이어주는 장치 – Continuation
이를 활용하는 프로그래밍스타일 – Continuation Passing Style

Sync의 장점 + Async의 장점

sync로직으로 async를 사용할 수 있게 함

하지만 sync flow가 어긋나므로 이전 sync flow의 상태를 기억하여 이어줄 장치 필요

상태를 기억하고 이어주는 장치 – Continuation
이를 활용하는 프로그래밍스타일 – Continuation Passing Style

Generator, Async, Asynchrous Iterators

Non Blocking For

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
};
```

```
nbFor(100, 10, working);
```

```
const working =_=>{};
for(let i = 0; i < 100000; i++) working();

const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) timeout(f, 0);
    };
    timeout(f, 0);
};
```

```
nbFor(100, 10, working);
```

Generator

```
const infinity = (function*(){
    let i = 0;
    while(true) yield i++;
})();
console.log(infinity.next());
```

```
const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) timeout(f, 0);
    };
    timeout(f, 0);
};
```

```
const nbFor = (max, load, block)=>{
    let i = 0;
    const f = time=>{
        let curr = load;
        while(curr-- && i < max){
            block();
            i++;
        }
        console.log(i);
        if(i < max - 1) timeout(f, 0);
    };
    timeout(f, 0);
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const nbFor = (max, load, block)=>{
  const iterator = gene(max, load, block);
  const f =_=>iterator.next().done||timeout(f);
  timeout(f, 0);
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const nbFor = (max, load, block)=>{
  const iterator = gene(max, load, block);
  const f =_=>iterator.next().done||timeout(f);
  timeout(f, 0);
};

nbFor(100, 10, working);
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

Promise

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene2 = function*(max, load, block) {
    let i = 0;
    while (i < max) {
        yield new Promise(res => {
            let curr = load;
            while (curr-- && i < max){
                block();
                i++;
            }
            console.log(i);
            timeout(res, 0);
        });
    }
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene2 = function*(max, load, block) {
    let i = 0;
    while (i < max) {
        yield new Promise(res => {
            let curr = load;
            while (curr-- && i < max){
                block();
                i++;
            }
            console.log(i);
            timeout(res, 0);
        });
    }
};
```

```
const nbFor = (max, load, block)=>{
  const iterator = gene(max, load, block);
  const f =_=>iterator.next().done||timeout(f);
  timeout(f, 0);
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene2 = function*(max, load, block) {
    let i = 0;
    while (i < max) {
        yield new Promise(res => {
            let curr = load;
            while (curr-- && i < max){
                block();
                i++;
            }
            console.log(i);
            timeout(res, 0);
        });
    }
};
```

```
const nbFor = (max, load, block)=>{
    const iterator = gene2(max, load, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
const gene = function*(max, load, block){
    let i = 0, curr = load;
    while(i < max){
        if(curr--){
            block();
            i++;
        }else{
            curr = load;
            console.log(i);
            yield;
        }
    }
};
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

Promise Jobs

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
nbFor(10000, _=>console.log("nb"));
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

Promise Jobs

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
nbFor(10000, _=>console.log("nb"));
const f = _=>{
  console.log("f");
  requestAnimationFrame(f);
};
requestAnimationFrame(f);
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

Promise Jobs

Frames

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
nbFor(10000, _=>console.log("nb"));
const f = _=>{
  console.log("f");
  requestAnimationFrame(f);
};
requestAnimationFrame(f);
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
nbFor(10000, _=>console.log("nb"));
const f = _=>{
  console.log("f");
  requestAnimationFrame(f);
};
requestAnimationFrame(f);
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    time(res);
  });
};
```

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>next(iterator.next()));
    next(iterator.next());
};
```

```
nbFor(10000, _=>console.log("nb"));
const f = _=>{
  console.log("f");
  requestAnimationFrame(f);
};
requestAnimationFrame(f);
```

```
const gene2 = function*(max, block) {
  let i = 0;
  while (i++ < max) yield new Promise(res => {
    block();
    res();
  });
};
```

```
const nbFor = (max, block)=>{
    const iterator = gene2(max, block);
    const next = ({value, done})=> done || value.then(v=>time(_=>next(iterator.next())));
    next(iterator.next());
};
```