Yvo
Solutions

SOLID && FAST C++

Loïc Yvonnet

`#include <C++>`

www.yvo.solutions

SOLID principles

C++

New Acronym: FAST

Reminders about SOLID

Single Responsibility Principle

```
class griffin {
public:
    constexpr void fly() const noexcept;
    constexpr void roar() const noexcept;
    constexpr void run() const noexcept;
    ...
};
```

Griffin by Design Rails from the Noun Project

Single Responsibility Principle

```cpp
class eagle {
public:
    constexpr void fly() const noexcept;
};
```

```cpp
class lion {
public:
    constexpr void roar() const noexcept;
};
```

```cpp
class horse {
public:
    constexpr void run() const noexcept;
};
```

Open/Close Principle

```cpp
void apply_tactics() {
    helicopter h1;
    tank t1;

    ambush(h1, t1);
}
```

```cpp
constexpr void ambush(
    maneuverable auto& unit1,
    maneuverable auto& unit2) noexcept
{
    unit1.turn_left();
    unit1.go_straight();

    unit2.turn_right();
    unit2.go_straight();
}
```

Open/Close Principle

```
void apply_tactics() {
    helicopter h1;
    tank t1;

    ambush(h1, t1);
}
```

```
constexpr void ambush(
    maneuverable auto& unit1,
    maneuverable auto& unit2) noexcept
{
    unit1.turn_left();
    unit1.go_straight();

    unit2.turn_right();
    unit2.go_straight();
}
```

```
template <typename T>
concept maneuverable = requires(T unit) {
    unit.go_straight();
    unit.turn_right();
    unit.turn_left();
};
```

Liskov Substitution Principle

```cpp
class square : public rectangle {};
```

```cpp
template <typename TPose, typename TNum>
constexpr void double_width(rectangle<TPose, TNum> auto& rect) noexcept {
    const auto width = rect.width() * 2;
    rect.width(width);
}
```

Interface Segregation Principle

```cpp
template <typename T>
concept chimera = requires(T griffin) {
    griffin.fly();
    griffin.roar();
    griffin.run();
    ...
};
```

Griffin by Design Rails from the Noun Project

Dependency Inversion Principle

```cpp
class microservice {
    http_client transport{"https://web.api.v1", 443};
    yaml_formatter format;
    database persistence{"Data Source=:memory:", "DB"};

public:
    void process(std::string_view data) {
        const auto response = send_request(data);                  // use transport
        const auto [key, value] = deserialize(response.body); // use format
        store(key, value);                                                  // use persistence
    }
};
```

Dependency Inversion Principle

```cpp
template <
    concepts::http_client<http::request, http::response> TTransport = http_client,
    concepts::yaml_formatter<yaml::object> TFormatter = yaml_formatter,
    concepts::database<db::result> TPersistence = database
>
class microservice {
    TTransport transport;
    TFormatter format;
    TPersistence persistence;

public:
    explicit microservice(const TTransport& t, const TFormatter& f, const TPersistence& p) :
        transport{t}, format{f}, persistence{p} {}

    void process(std::string_view data) {
        const auto response = send_request(data);
        const auto [key, value] = deserialize(response.body);
        store(key, value);
    }
};
```

Dependency Inversion Principle

```cpp
void process_data() {
    // Register
    http_client transport{"https://web.api.v1", 443};
    yaml_formatter format;
    database persistence{"Data Source=:memory:", "DB"};

    // Resolve
    microservice srv(transport, format, persistence);

    // Use
    srv.process("data");
}
```

F
A
S
T

#include <C++>

Functional style

A

S

T

#include <C++>

Functional Style

Immutability	Purity(ish)
const constexpr	constexpr consteval TMP

Functional Style

```cpp
template <auto N>
void repeat_n(std::invokable<void(decltype(N))> auto&& f) {
    ranges::for_each(ranges::view::iota(0, N), [f](auto i) {
        f(i);
    });
}

void test_repeat() {
    repeat_n<10>([](int i) {
        std::cout << "Repeat - " << i << '\n';
    });
}
```

F
A
S
T

#include <C++>

F

Algorithms

S

T

Algorithms

- Write code in terms of algorithms and data structures.
- No raw loops.

F A S T

#include <C++>

F

A

Security

T

#include <C++>

Security

- Don’t use insecure APIs (e.g. std::gets - removed from C++14).
- Check all inputs.
- Use static analysers.
- Follow best practices (hash & salt, strong encryption, certificates, etc.).
- Follow standards (e.g. OWASP, MISRA, AUTOSAR, etc.).
- Get your code audited by experts.

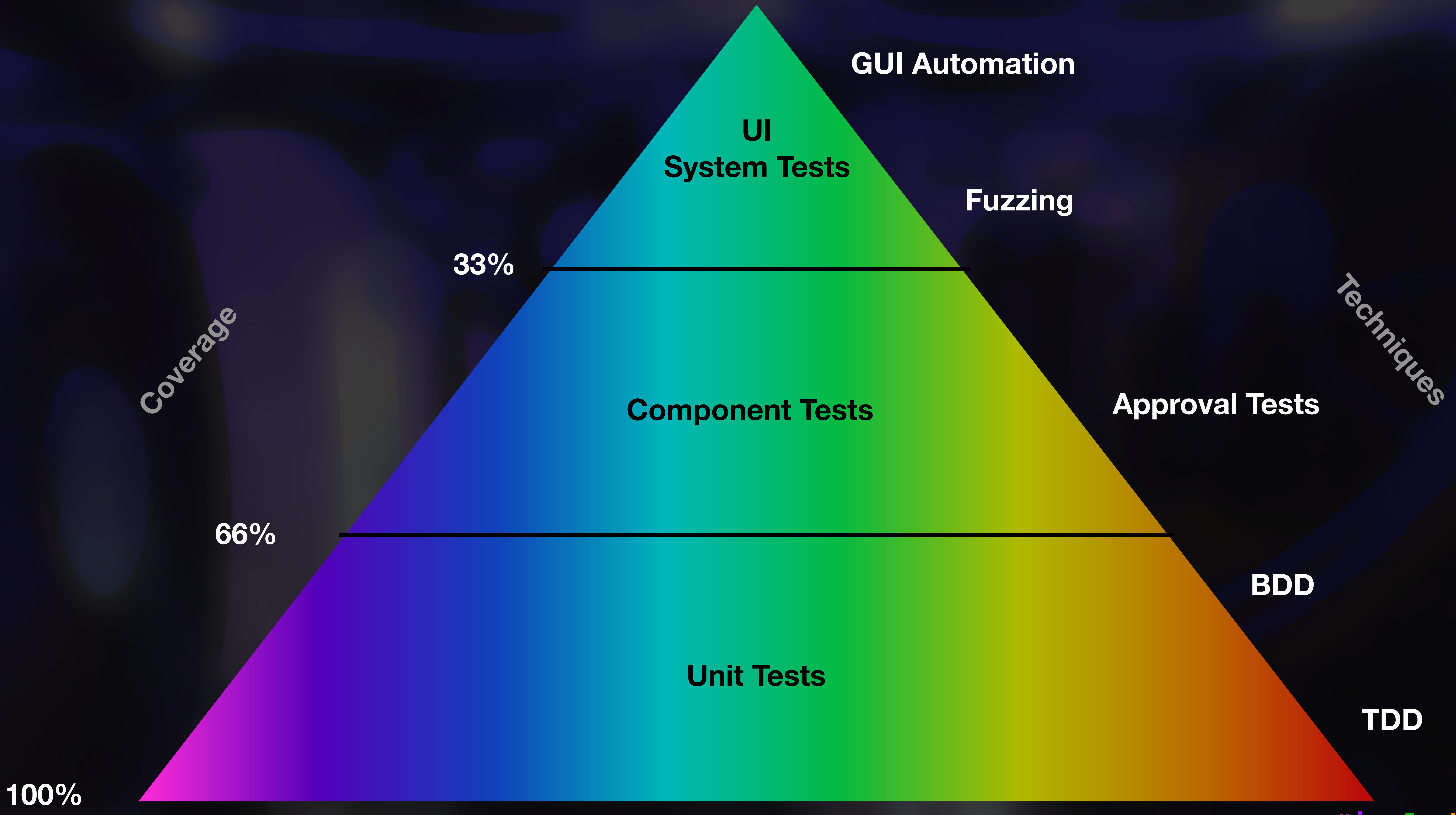
F A S T

#include <C++>

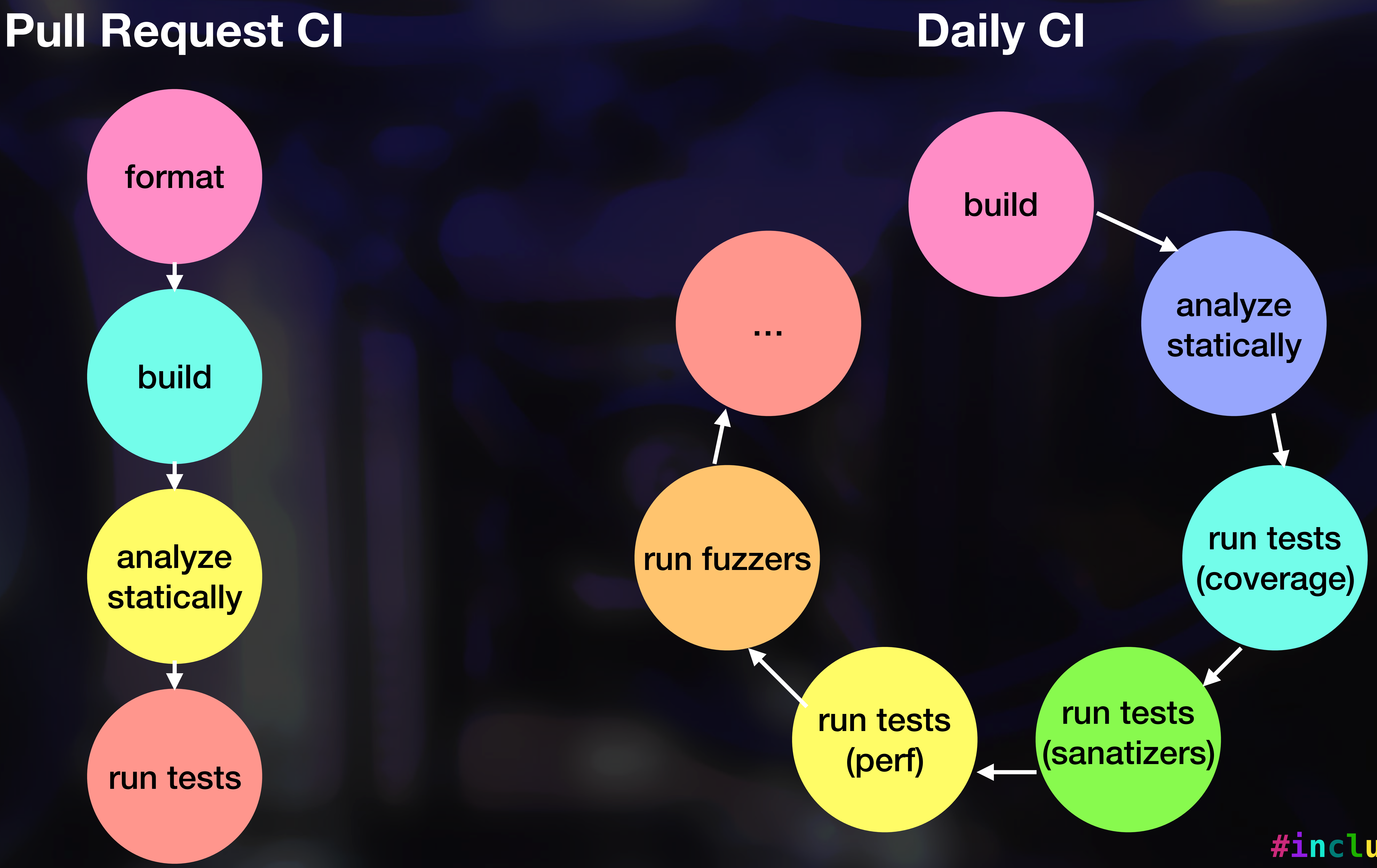
F
A
S
Tests

#include <C++>

Tests Pyramid



Continuous Integration



Wait... What?

Yvo
Solutions

www.yvo.solutions

Thank you

@lyvonnet

loic-yvonnet

lyvonnet

#include <C++>