LAPP
Laboratoire d'Annecy de Physique des Particules

Introduction : Compilation

Pierre Aubert

ESCAPE
cta cherenkov telescope array
cnrs
UNIVERSITÉ SAVOIE MONT BLANC

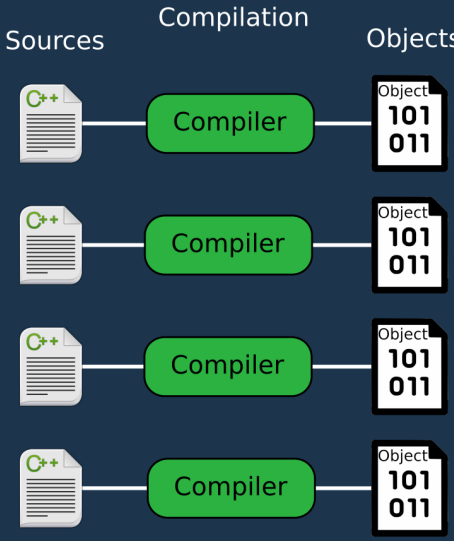
Compilation : From Text to Binary

Compilation : From Text to Binary

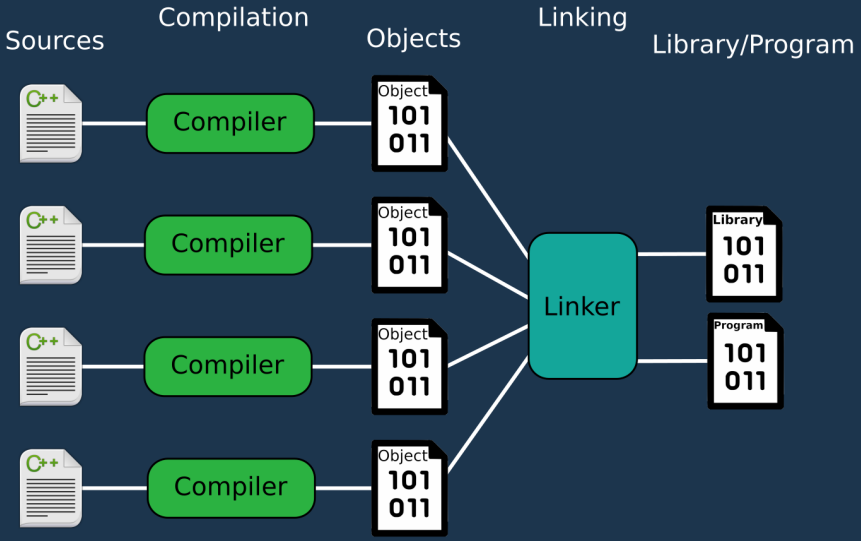
Sources



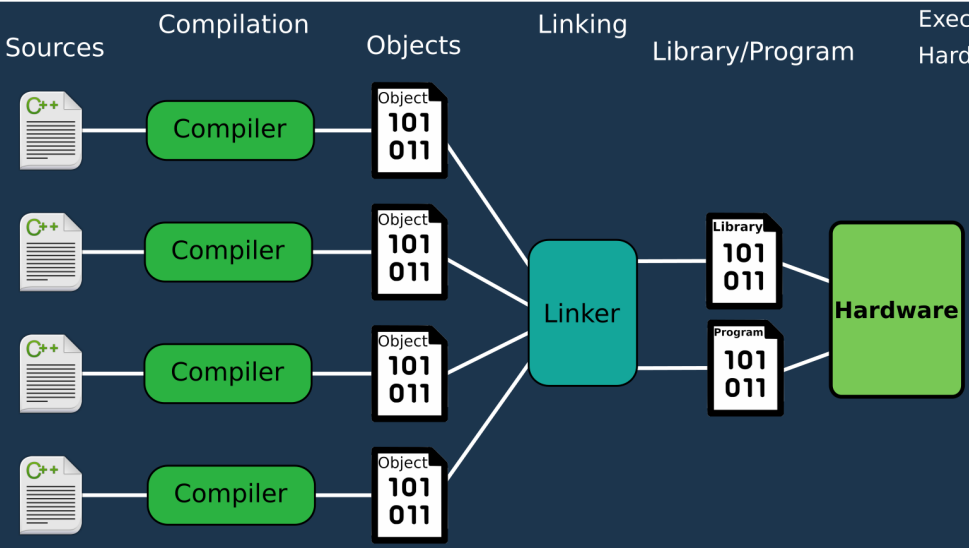

Compilation : From Text to Binary



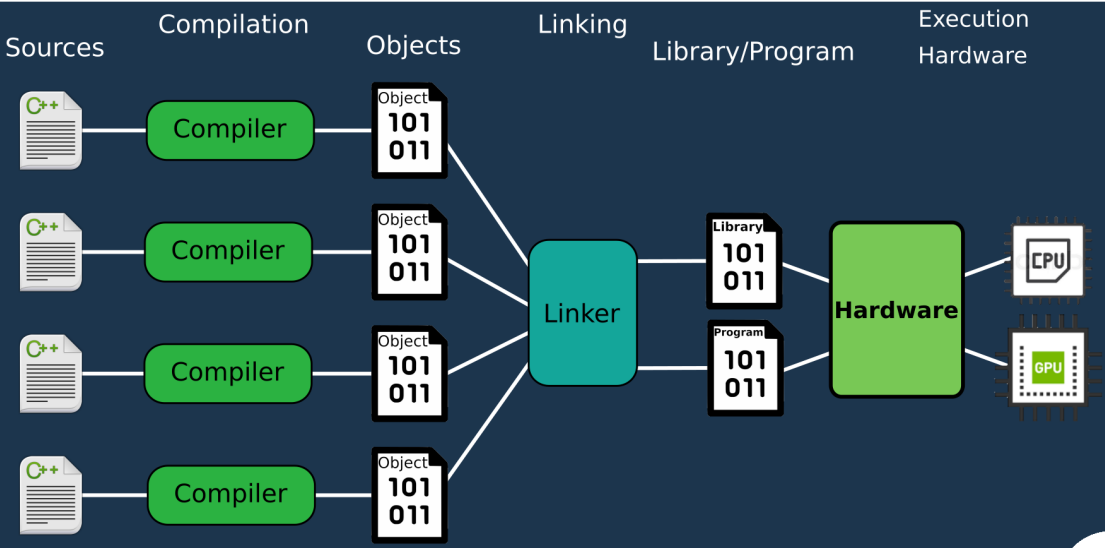
Compilation : From Text to Binary



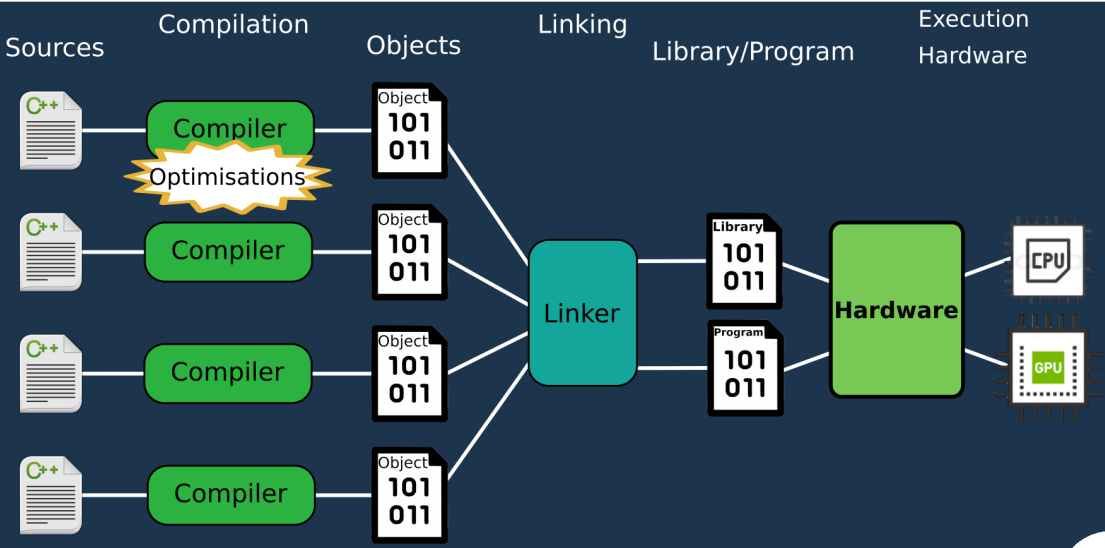
Compilation : From Text to Binary



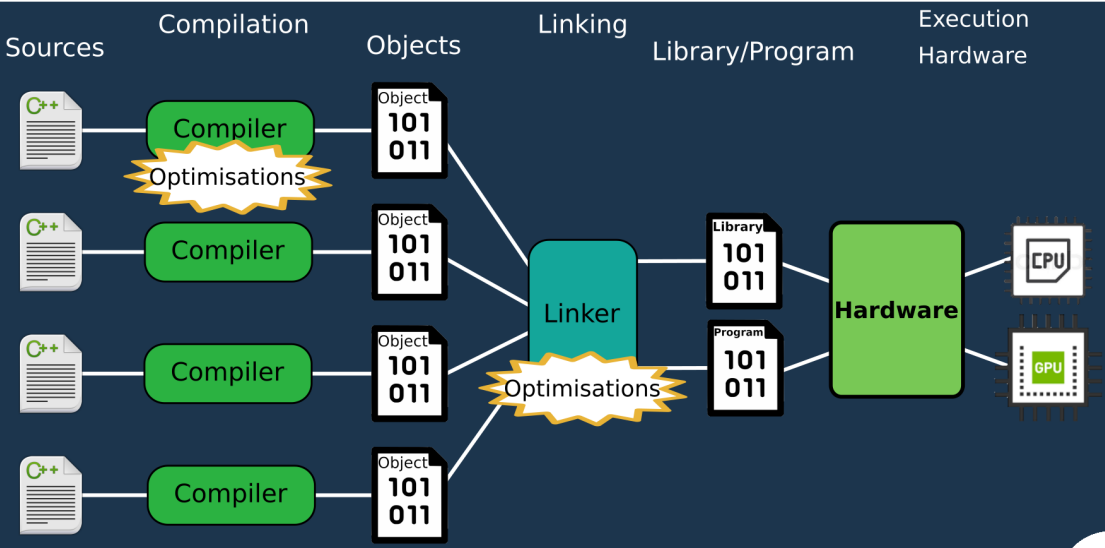
Compilation : From Text to Binary



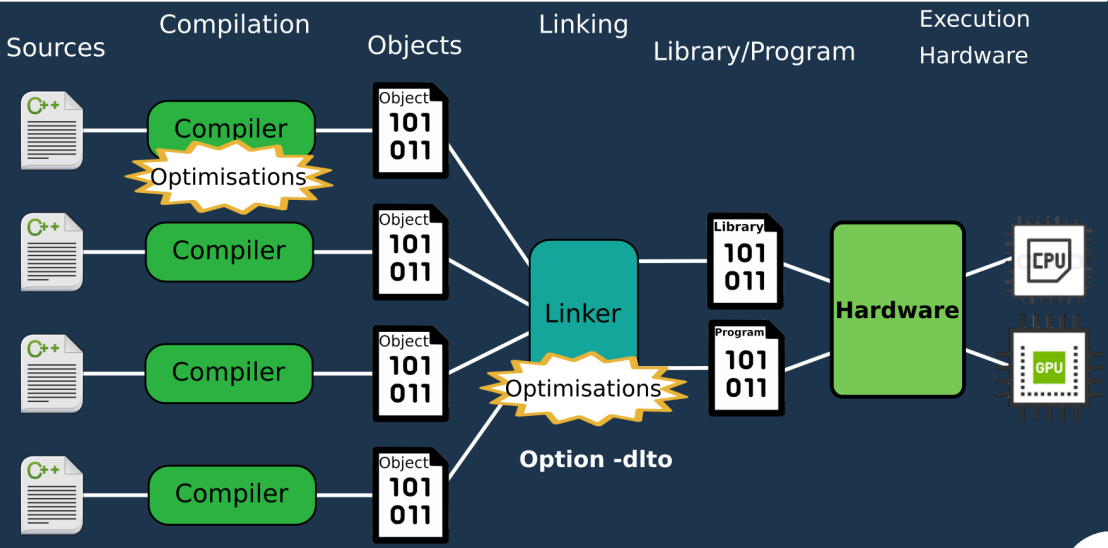
Compilation : From Text to Binary



Compilation : From Text to Binary



Compilation : From Text to Binary



Compiler workflow

Compiler workflow

Languages

Fortran

Compiler workflow

Languages

Fortran

Binaries

Compiler workflow

Languages

Front-End

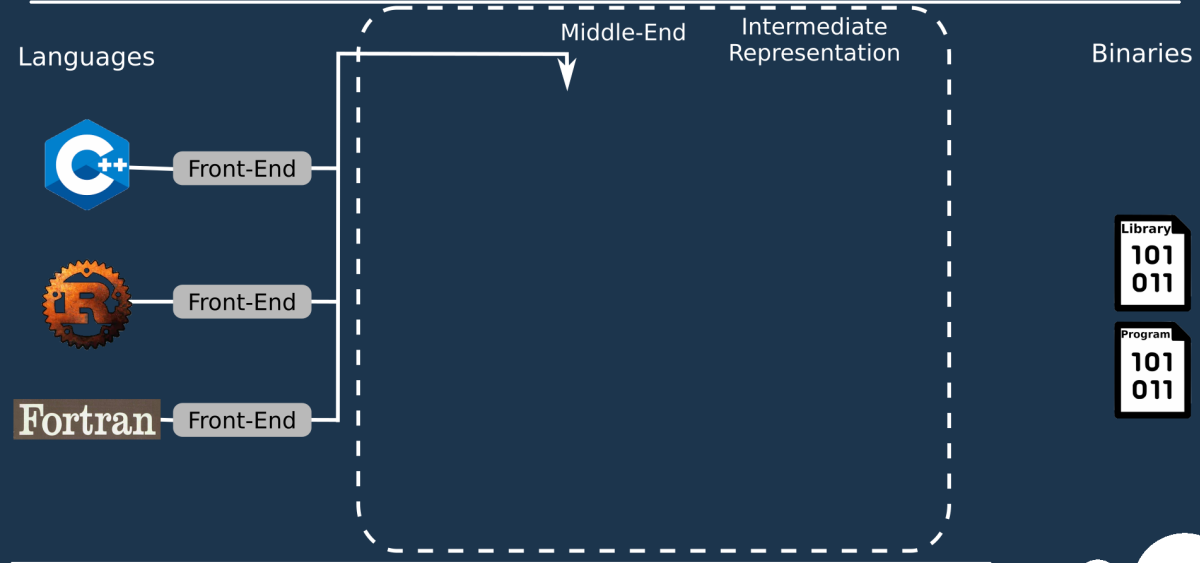
Front-End

Fortran

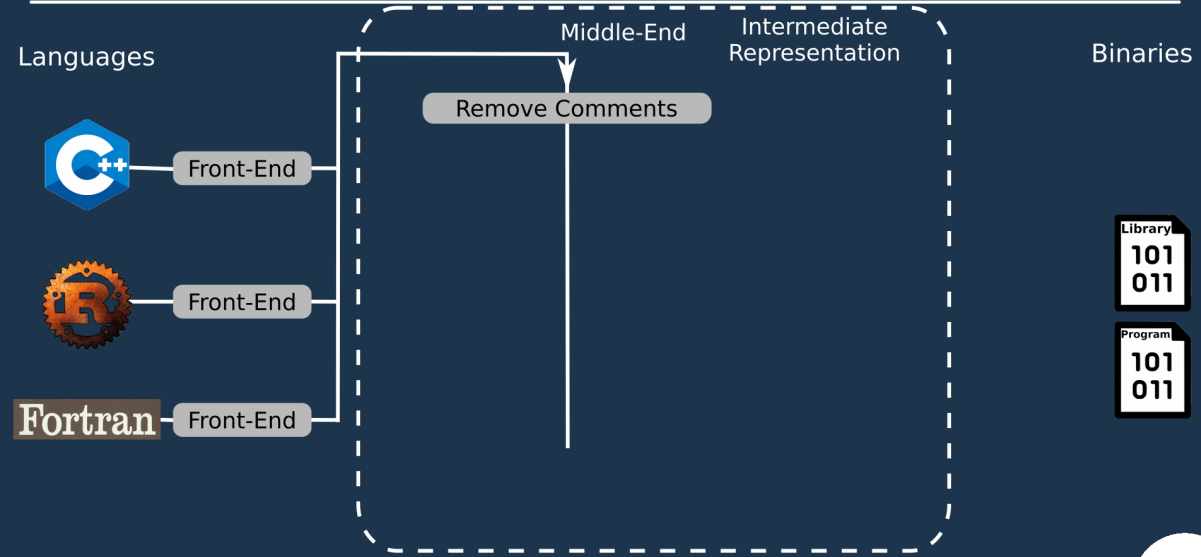
Front-End

Binaries

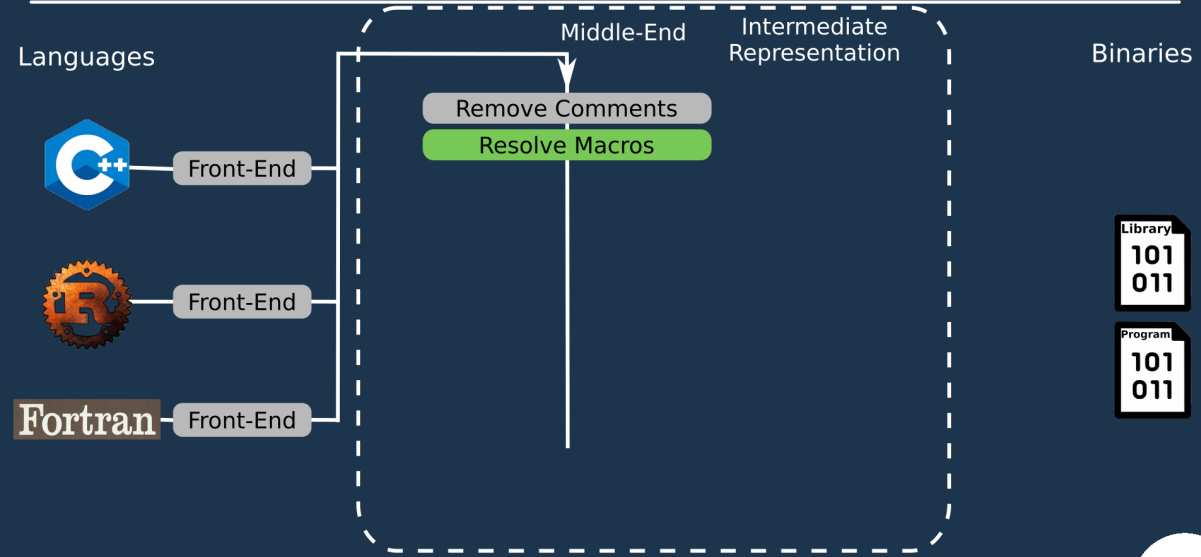
Compiler workflow



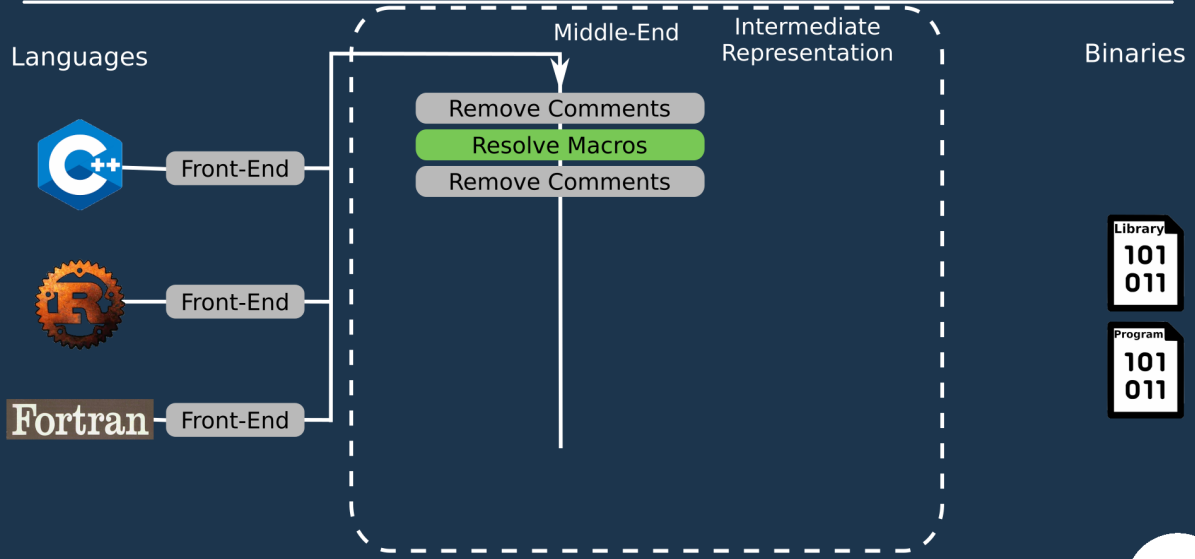
Compiler workflow



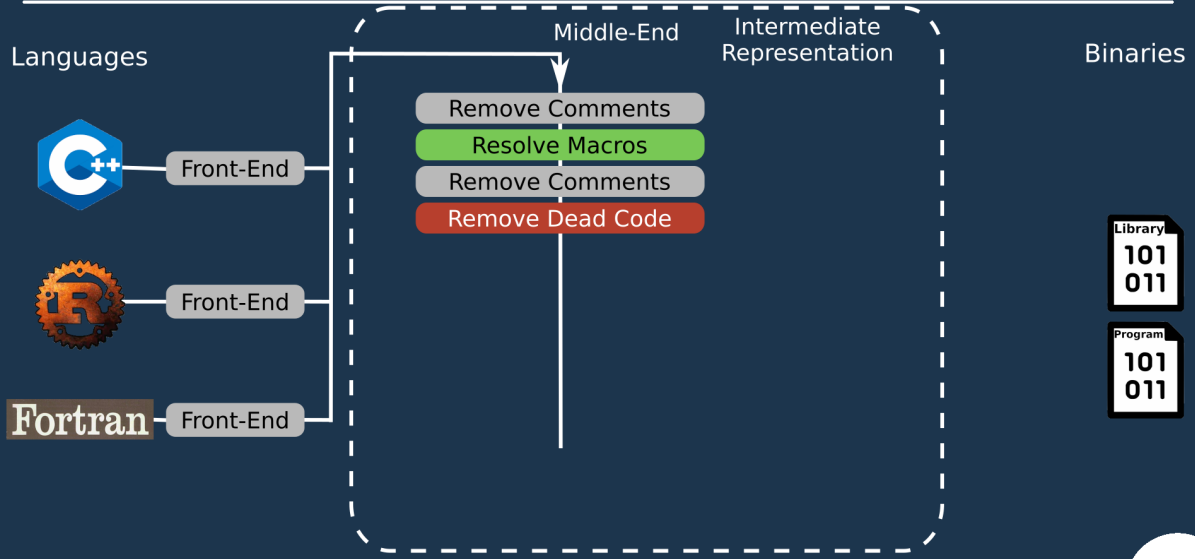
Compiler workflow



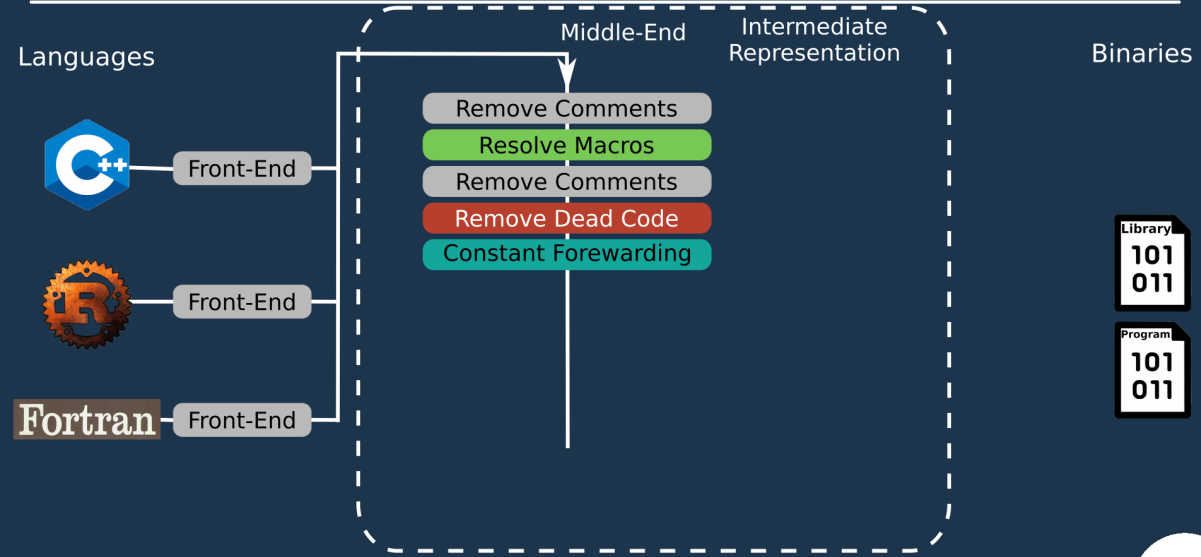
Compiler workflow



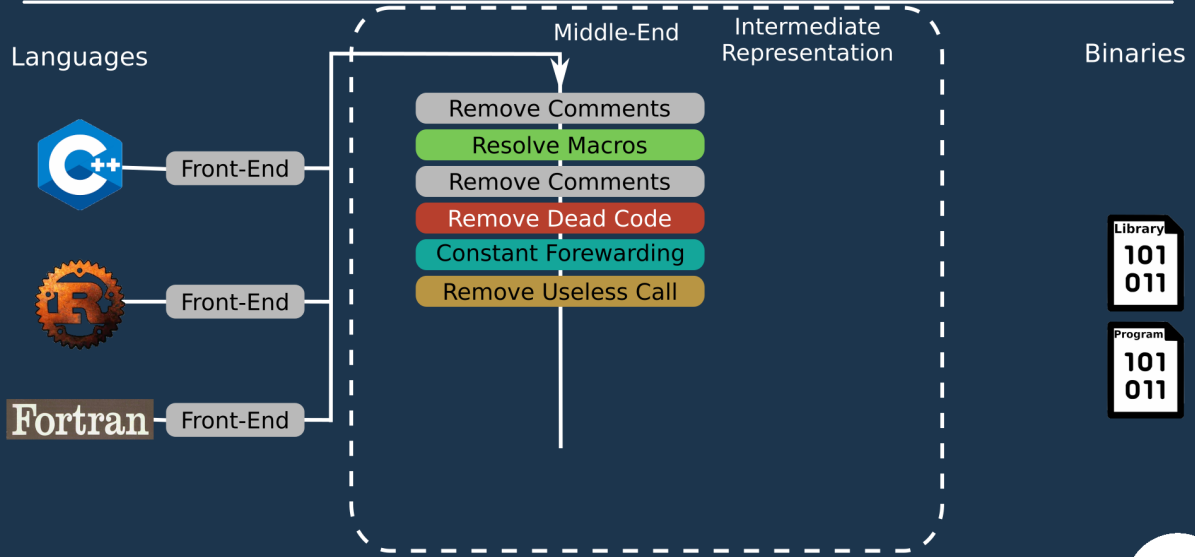
Compiler workflow



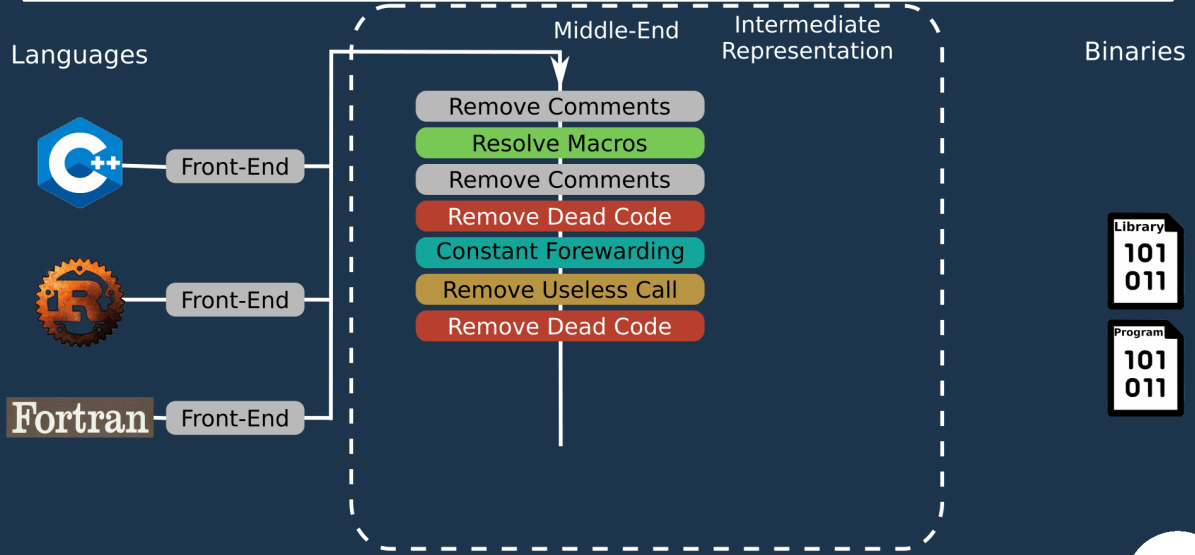
Compiler workflow



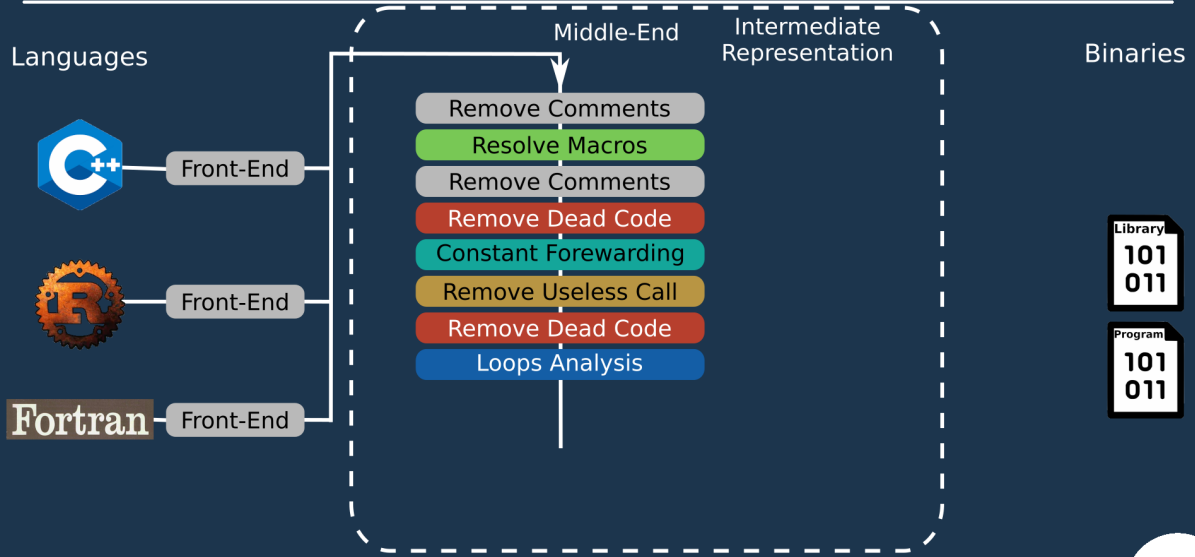
Compiler workflow



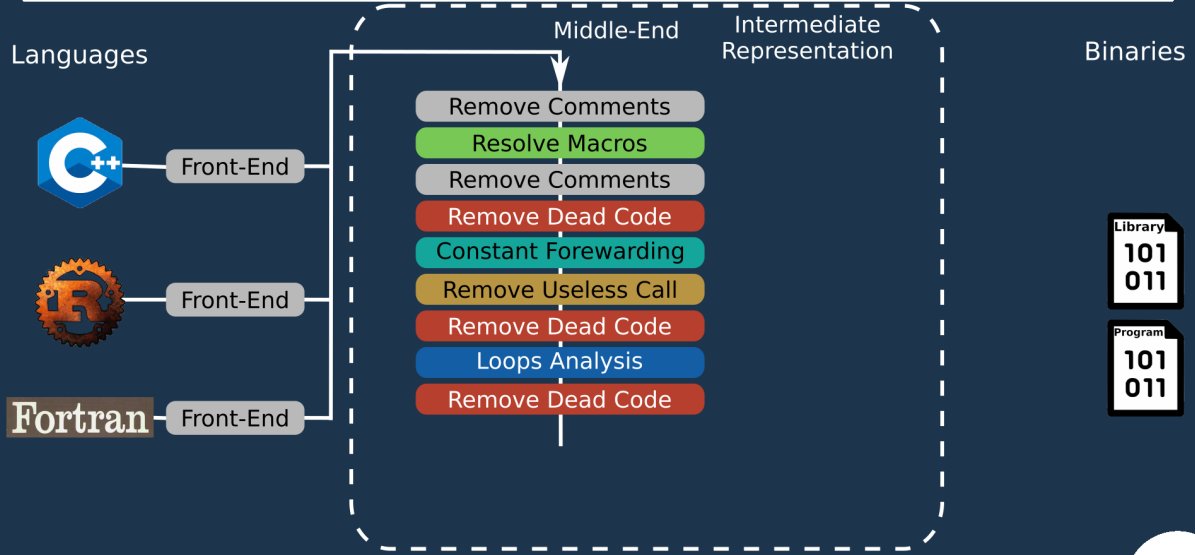
Compiler workflow



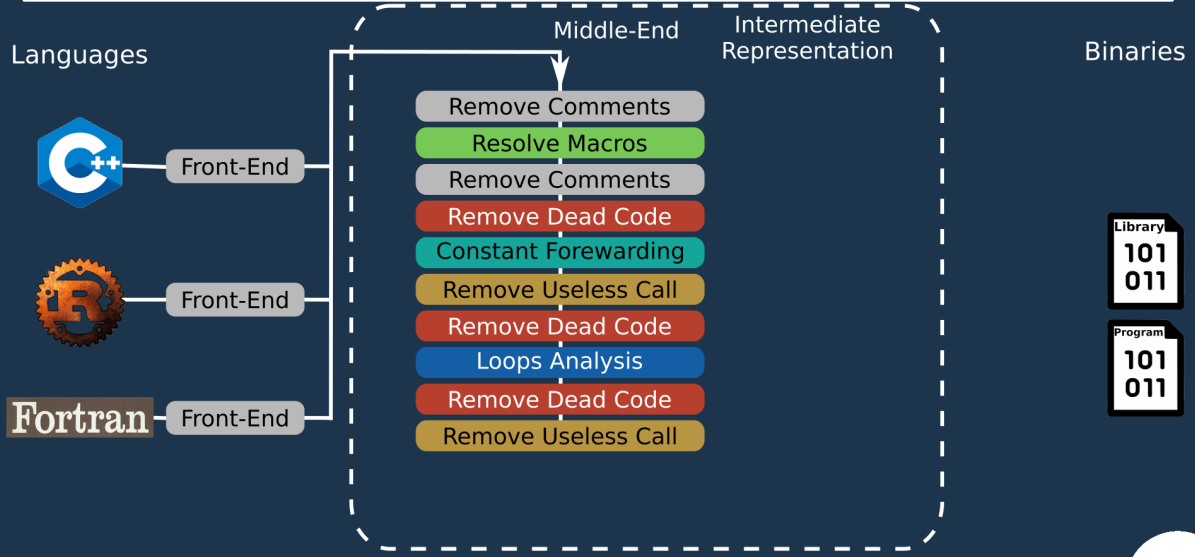
Compiler workflow



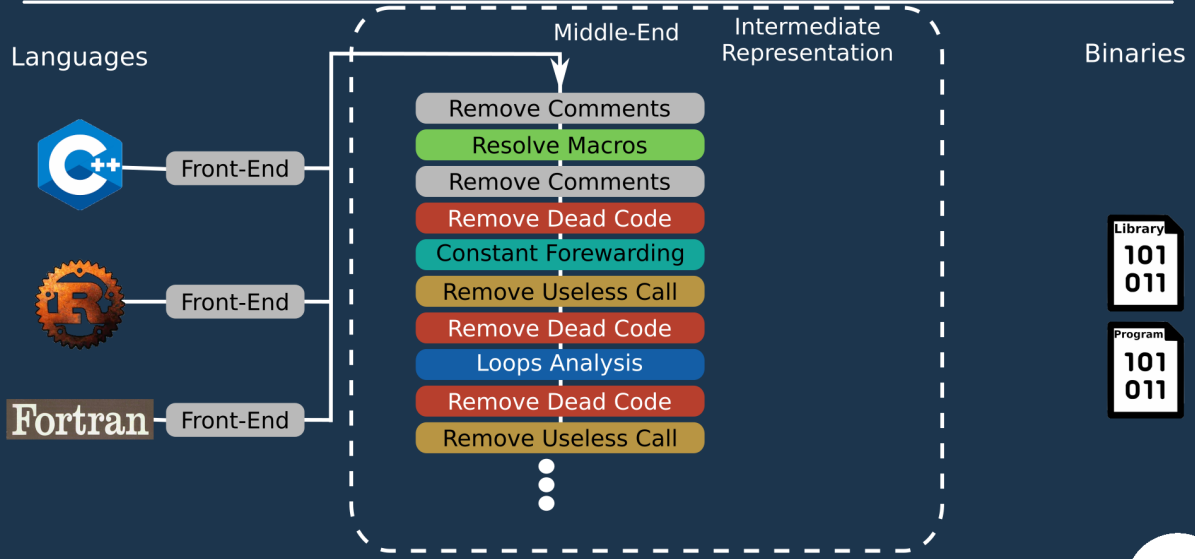
Compiler workflow



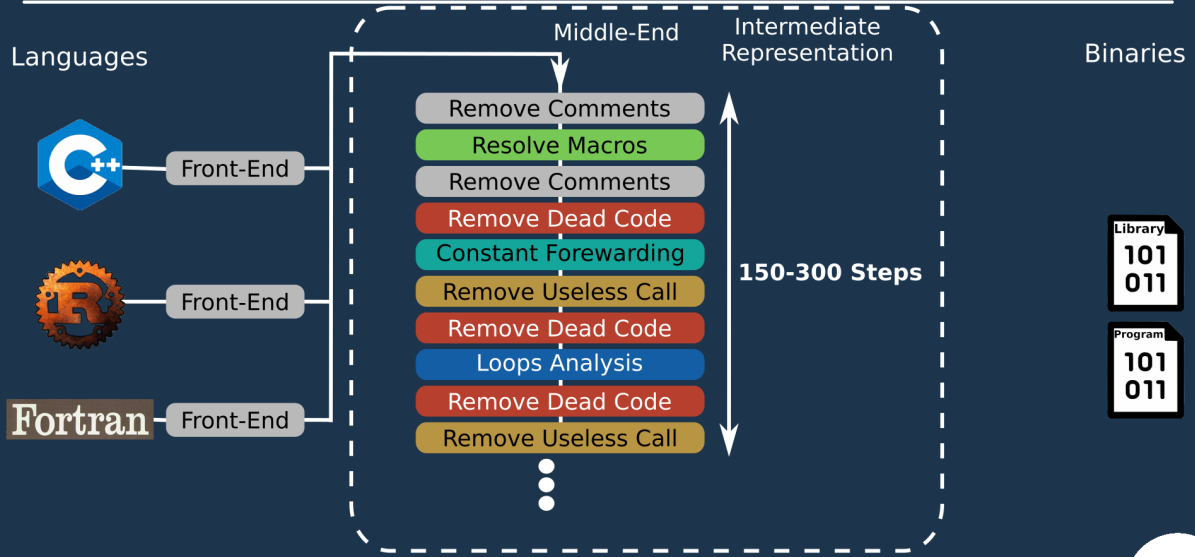
Compiler workflow



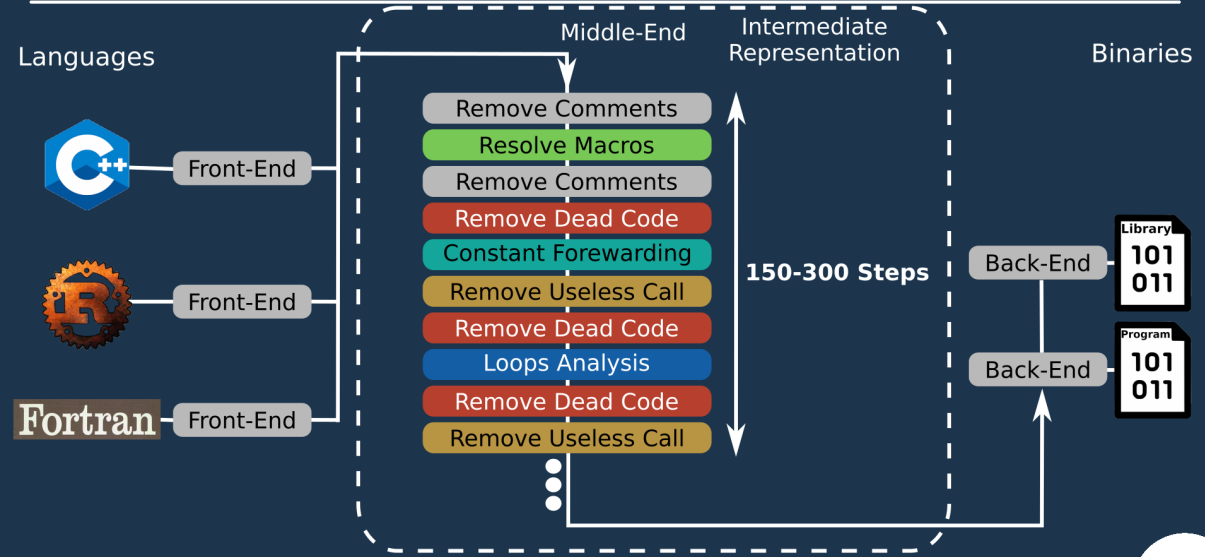
Compiler workflow



Compiler workflow



Compiler workflow



Compilation options

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Compilation options

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

- **-O0** : default
  - Try to reduce compilation time, but **-Og** is better for debugging.

Compilation options

`https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`

- **-O0** : default
  - Try to reduce compilation time, but **-Og** is better for debugging.
- **-O1**
  - Constant forewarding, remove dead code (never called code)...

Compilation options

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

- **-O0** : default
  - Try to reduce compilation time, but **-Og** is better for debugging.
- **-O1**
  - Constant forewarding, remove dead code (never called code)...
- **-O2**
  - Partial function inlining, Assume strict aliasing...

Compilation options

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

- **-O0** : default
  - Try to reduce compilation time, but **-Og** is better for debugging.
- **-O1**
  - Constant forewarding, remove dead code (never called code)...
- **-O2**
  - Partial function inlining, Assume strict aliasing...
- **-O3**
  - More function inlining, loop unrolling, partial vectorization...

Compilation options

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

- **-O0** : default
  - Try to reduce compilation time, but **-Og** is better for debugging.
- **-O1**
  - Constant forewarding, remove dead code (never called code)...
- **-O2**
  - Partial function inlining, Assume strict aliasing...
- **-O3**
  - More function inlining, loop unrolling, partial vectorization...
- **-Ofast**
  - Disregard strict standards compliance. Enable **-ffast-math**, stack size is hardcoded to 32 768 bytes (borrowed from **gfortran**). **Possibly degrades the computation accuracy.**

Examples

Examples

```
float square(float x){
»       return x*x;
}
int main(){
»       float y(42.0f);
»       std::cout << square(y) << std::endl;
»       return 0;
}
```

Examples

```
float square(float x){
»       return x*x;
}
int main(){
»       float y(42.0f);
»       std::cout << square(y) << std::endl;
»       return 0;
}
```

Constant forewarding

```
float square(float x){
»       return x*x;
}
int main(){
»       std::cout << square(42.0f) << std::endl;
»       return 0;
}
```

Examples

```
float square(float x){
»       return x*x;
}
int main(){
»       float y(42.0f);
»       std::cout << square(y) << std::endl;
»       return 0;
}
```

Constant forewarding

```
float square(float x){
»       return x*x;
}
int main(){
»       std::cout << square(42.0f) << std::endl;
»       return 0;
}
```

Function inlining

```
int main(){
»       std::cout << 42.0f*42.0f << std::endl;
»       return 0;
}
```

Examples

```
float square(float x){
»       return x*x;
}
int main(){
»       float y(42.0f);
»       std::cout << square(y) << std::endl;
»       return 0;
}
```

Constant forewarding

```
float square(float x){
»       return x*x;
}
int main(){
»       std::cout << square(42.0f) << std::endl;
»       return 0;
}
```

Function inlining

```
int main(){
»       std::cout << 42.0f*42.0f << std::endl;
»       return 0;
}
```

Constant expression solving

```
int main(){
»       std::cout << 1764.0f << std::endl;
»       return 0;
}
```

Examples

```
float square(float x){
»       return x*x;
}
int main(){
»       float y(42.0f);
»       std::cout << square(y) << std::endl;
»       return 0;
}
```

Constant forewarding

```
float square(float x){
»       return x*x;
}
int main(){
»       std::cout << square(42.0f) << std::endl;
»       return 0;
}
```

Function inlining

```
int main(){
»       std::cout << 42.0f*42.0f << std::endl;
»       return 0;
}
```

Constant expression solving

```
int main(){
»       std::cout << 1764.0f << std::endl;
»       return 0;
}
```

No more :
- **Function call**
- **Multiplication**
- **Variable declaration**

Loop Unrolling

Loop Unrolling

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; ++i){
		tabRes[i] = tabX[i]*tabY[i];
	}
}
```

Base

Loop Unrolling

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; ++i){
		tabRes[i] = tabX[i]*tabY[i];
	}
}
```

Base

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; i += 2lu){
		tabRes[i] = tabX[i]*tabY[i];
		tabRes[i + 1] = tabX[i + 1]*tabY[i + 1];
	}
}
```

Loop unrolled 2 times

Loop Unrolling

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; ++i){
		tabRes[i] = tabX[i]*tabY[i];
	}
}
```

Base

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; i += 2lu){
		tabRes[i] = tabX[i]*tabY[i];
		tabRes[i + 1] = tabX[i + 1]*tabY[i + 1];
	}
}
```

Increase Computing in Iteration

Loop unrolled 2 times

Loop Unrolling

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; ++i){
		tabRes[i] = tabX[i]*tabY[i];
	}
}
```

Base

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; i += 2lu){
		tabRes[i] = tabX[i]*tabY[i];
		tabRes[i + 1] = tabX[i + 1]*tabY[i + 1];
	}
}
```

Increase Computing in Iteration

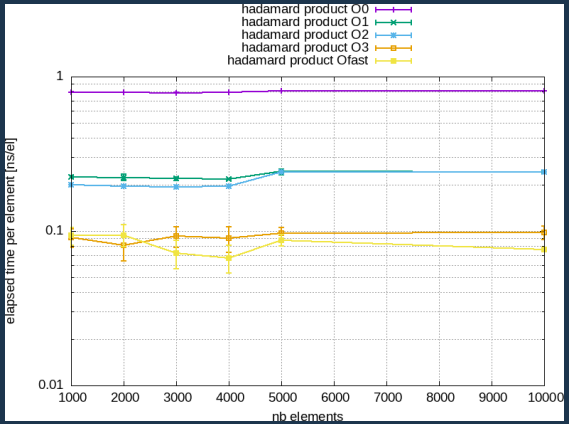
Loop unrolled 2 times

```
void hadamard(float* tabRes, const float * tabX, const float * tabY, size_t nbElement){
	for(size_t i(0lu); i < nbElement; i += 4lu){
		tabRes[i] = tabX[i]*tabY[i];
		tabRes[i + 1] = tabX[i + 1]*tabY[i + 1];
		tabRes[i + 2] = tabX[i + 2]*tabY[i + 2];
		tabRes[i + 3] = tabX[i + 3]*tabY[i + 3];
	}
}
```

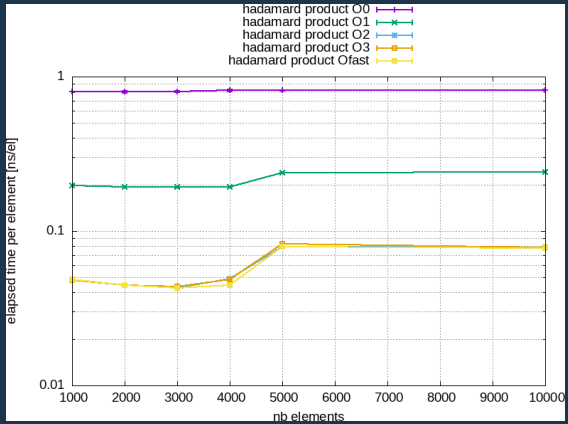
Loop unrolled 4 times

The Hadamard product : Basic Options

G++ 11

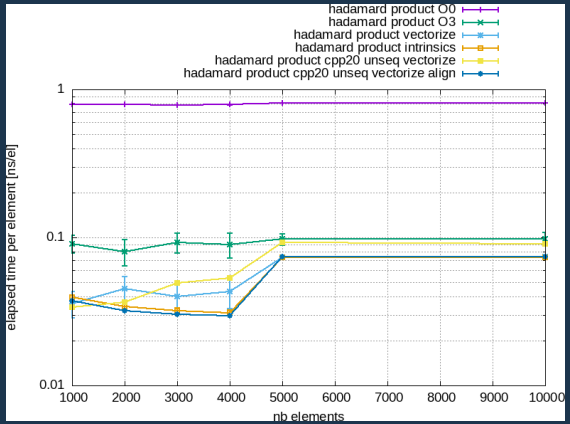


CLang++ 14

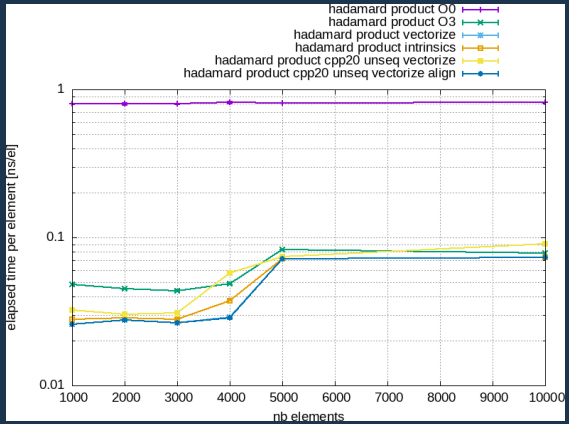


The Hadamard product : Vectorization

G++ 11



CLang++ 14



Conclusion

Conclusion

- Use **both** compilers
- Test **compilation options**
- Check with **Maqao** and **Valgrind**
- Check with **Performance Tests** (**MicroBenchmark**)