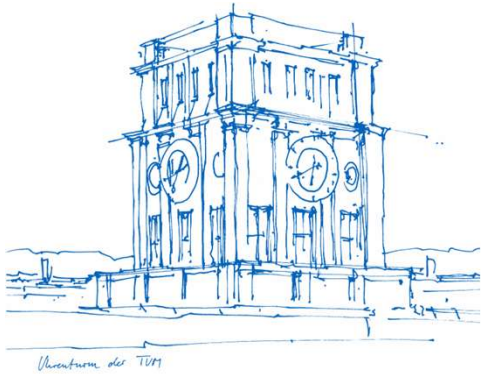
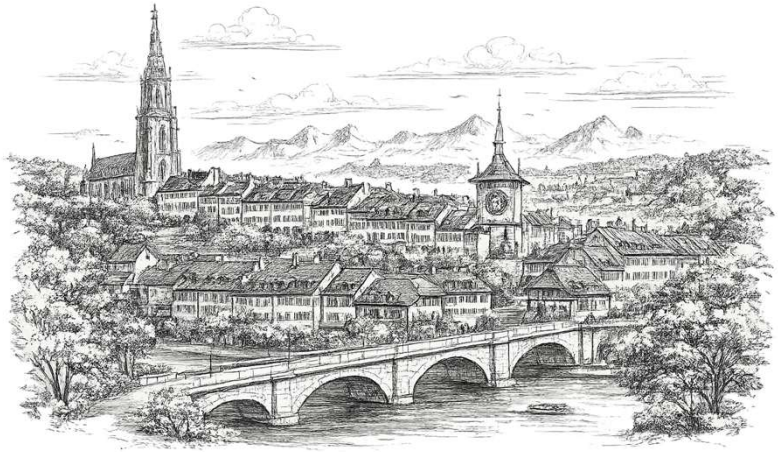


Classical Compiler Passes, Quantum Edition: Static Analyses for Quantum Programs

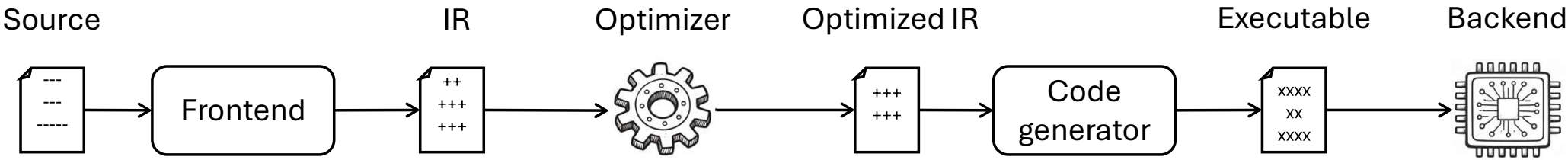
Technical University of Munich

Yanbin Chen, Yu Wang, Christian B. Mendl, Helmut Seidl

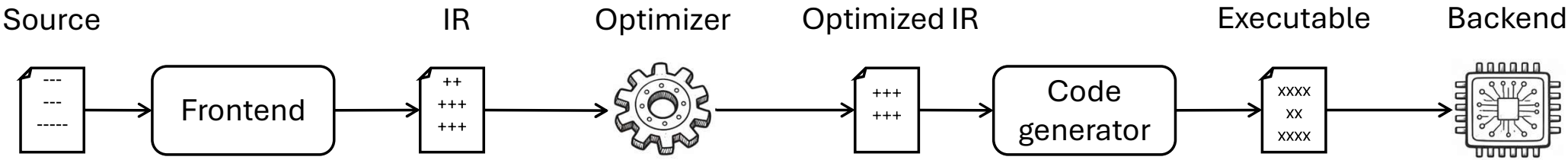


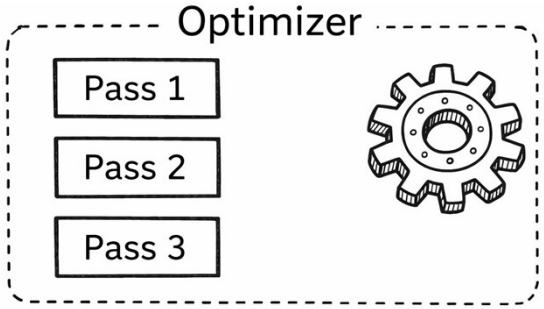
Q-STAV, Feb. 2026, Bern

Classical vs. quantum compiler

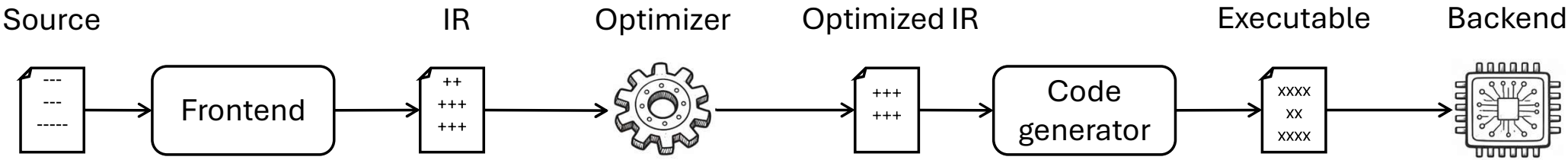


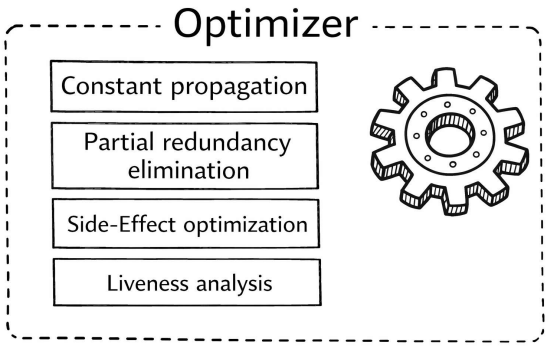
Classical vs. quantum compiler



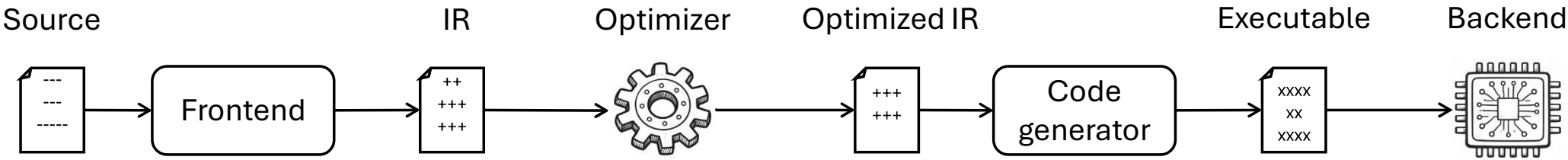


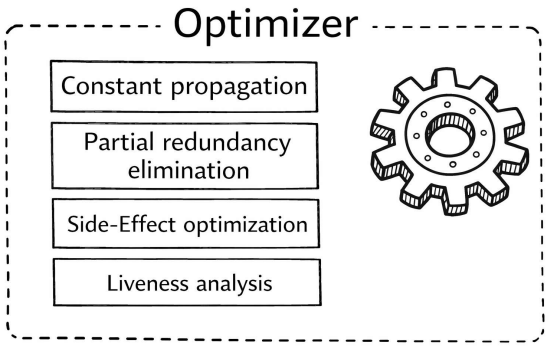
Classical vs. quantum compiler





Classical vs. quantum compiler



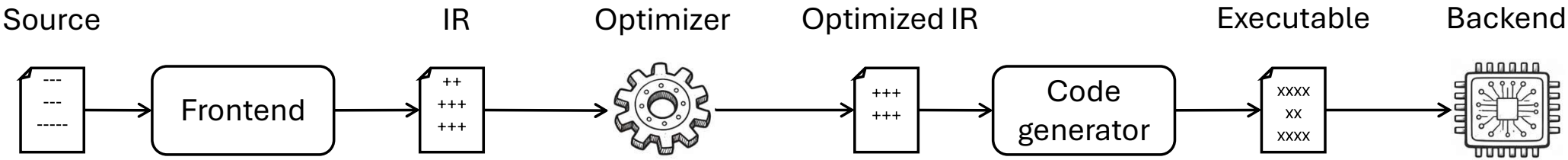


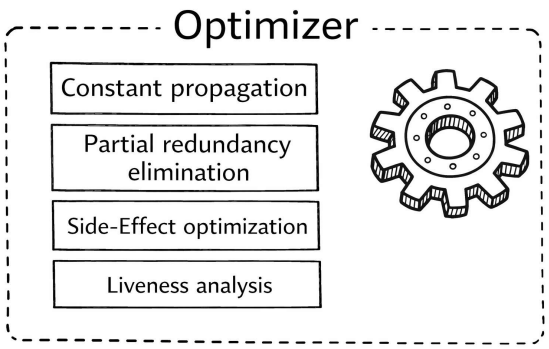

```
int a = 3;
int b = a * 4;
int c = b + 1;
return c;
```

CP

```
return 13;
```

Classical vs. quantum compiler






```
int a = 3;
int b = a * 4;
int c = b + 1;
return c;
```

CP

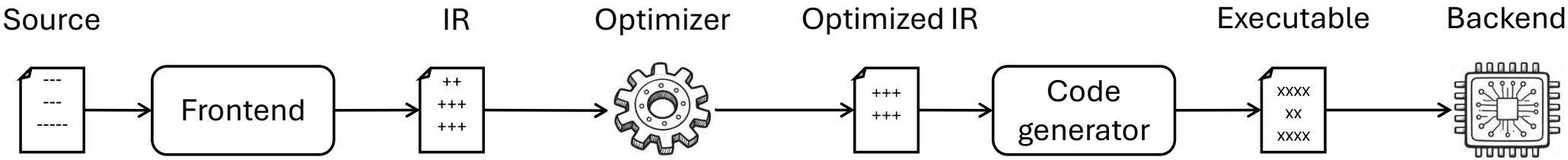
```
return 13;
```

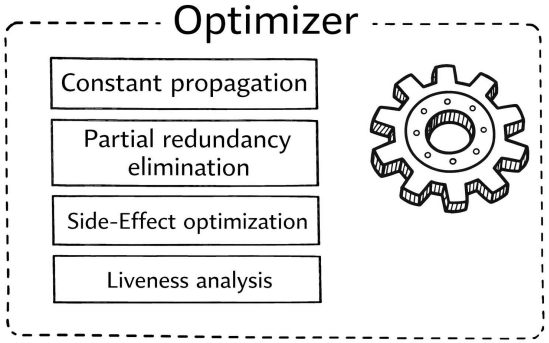
```
int t = f(a, b);
return a + 1;
```

LA

```
return a + 1;
```

Classical vs. quantum compiler






```
int a = 3;
int b = a * 4;
int c = b + 1;
return c;
```

CP

```
return 13;
```

```
int t = f(a, b);
return a + 1;
```

LA

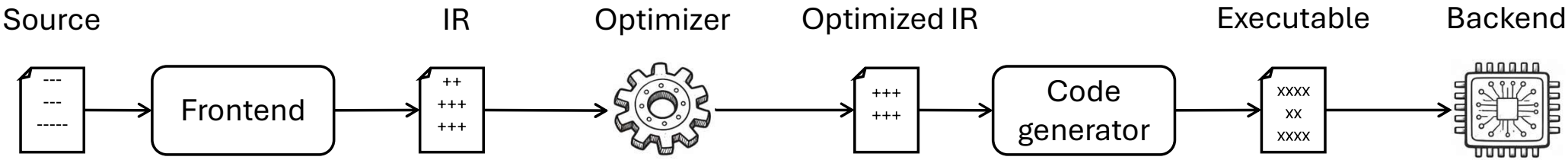
```
return a + 1;
```

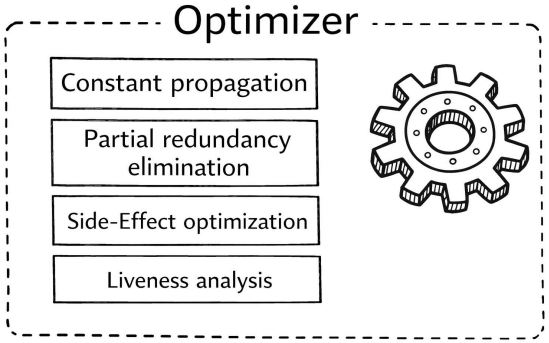
```
int t;
if (cond) {
   t = x + y;
   use(t);
}
int u = x + y;
use(u);
```

PRE

```
int t = x + y;
if (cond) {
   use(t);
}
use(t);
```

Classical vs. quantum compiler






```
int a = 3;
int b = a * 4;
int c = b + 1;
return c;
```

CP

```
return 13;
```

```
int t = f(a, b);
return a + 1;
```

LA

```
return a + 1;
```

```
int t;
if (cond) {
   t = x + y;
   use(t);
}
int u = x + y;
use(u);
```

PRE

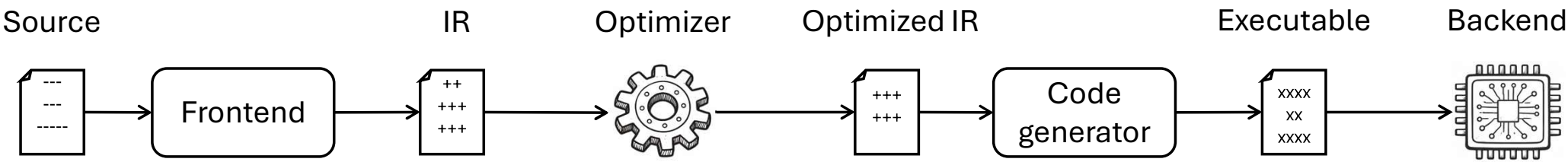
```
int t = x + y;
if (cond) {
   use(t);
}
use(t);
```

```
...
b = a-2;
if (cond)
   print(a-1);
else
   print(b+1);
...
```

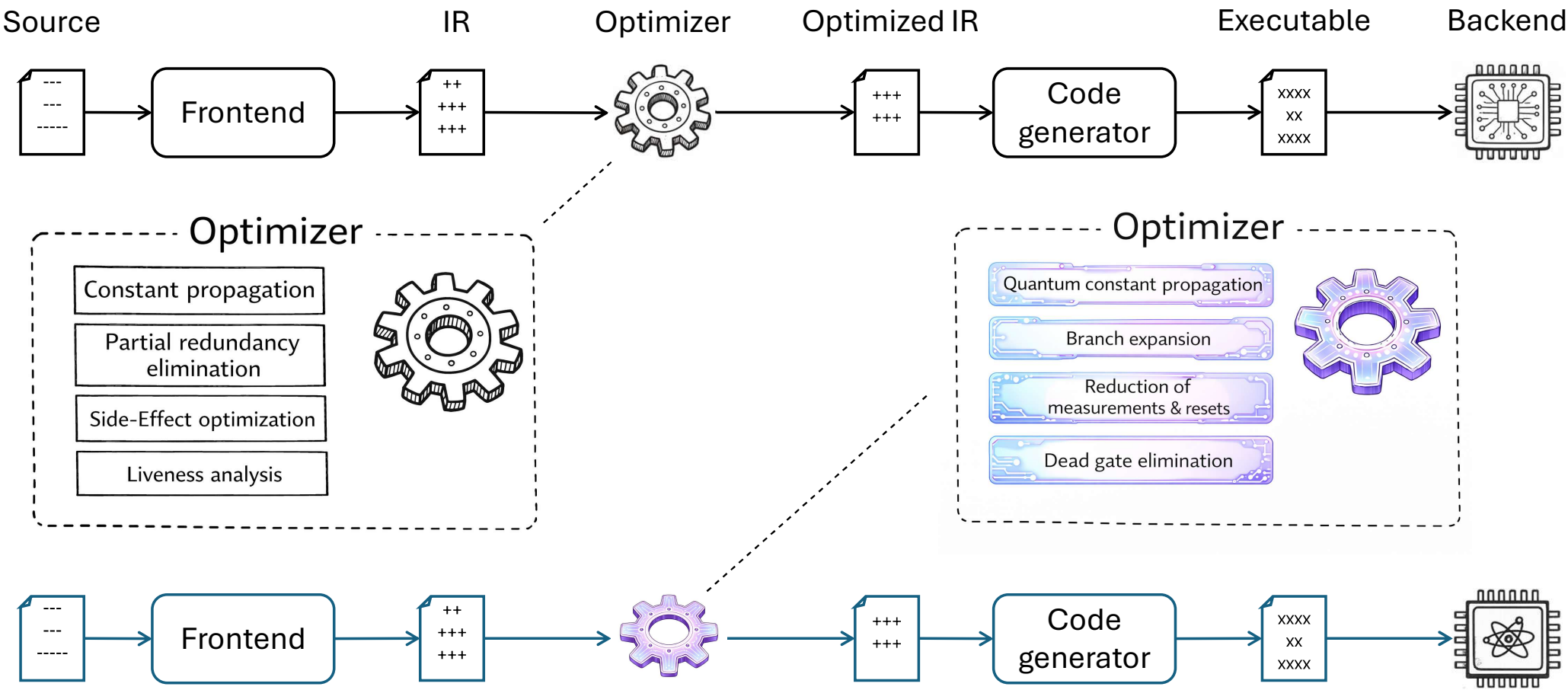
SEO

```
...
b=a-2
print(a-1);
...
```

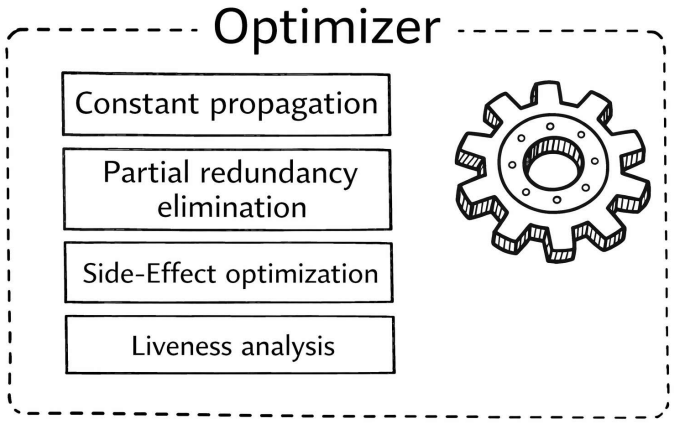
Classical vs. quantum compiler

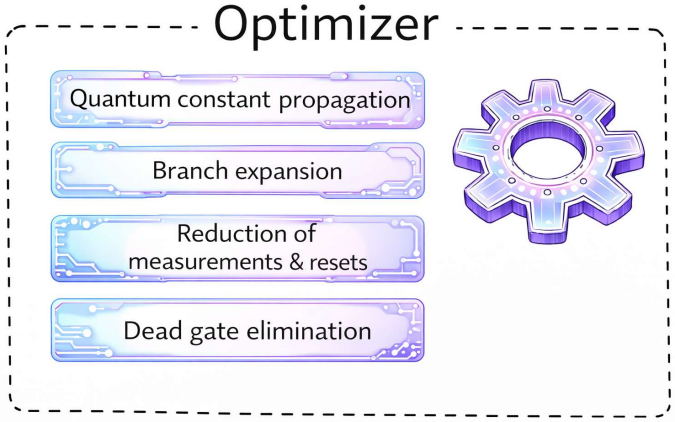


Classical vs. quantum compiler

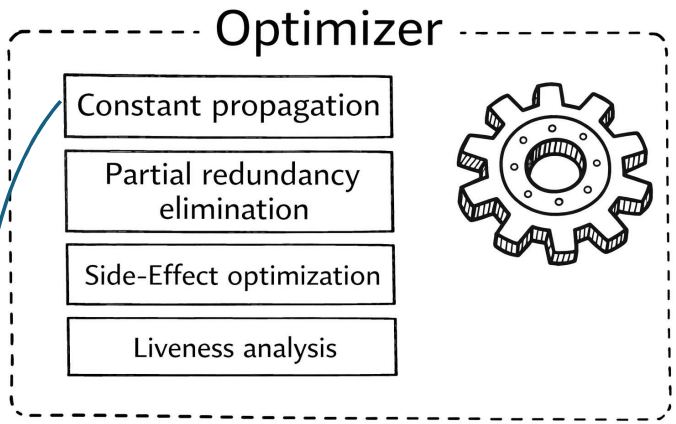


Classical vs. quantum compiler





Classical vs. quantum compiler

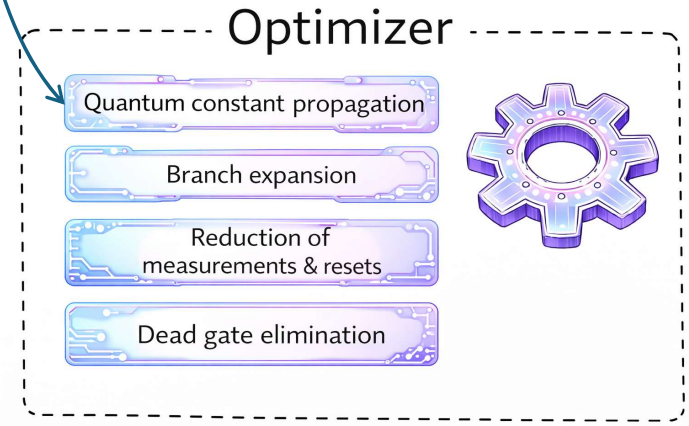



```
int a = 3;
int b = a * 4;
int c = b + 1;
return c;
```

CP

```
return 13;
```

Inspire

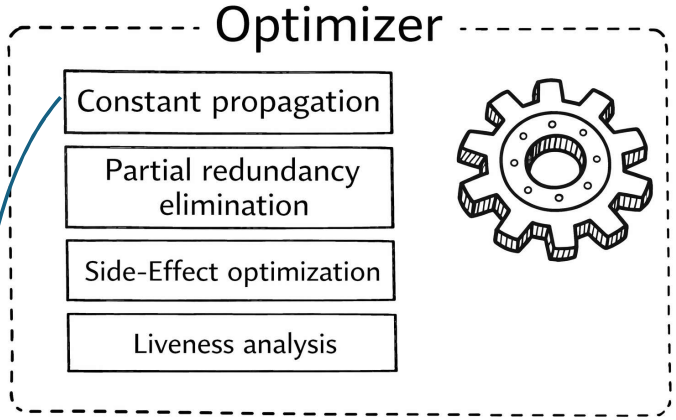



```
...
// whenever q0 in |1⟩, q1 is in |1⟩
ccx q0, q1, q2;
...
```

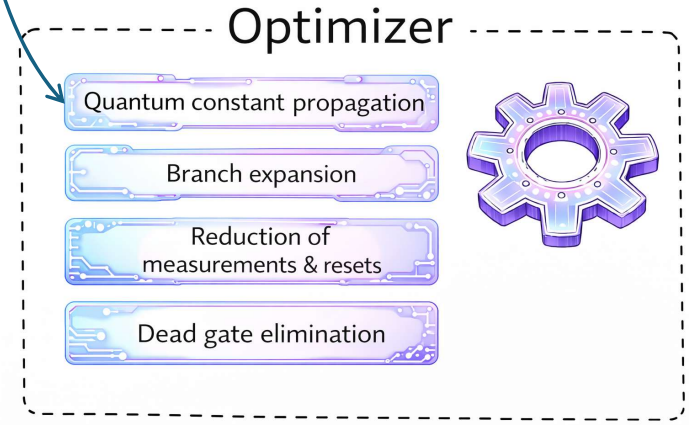
QCP

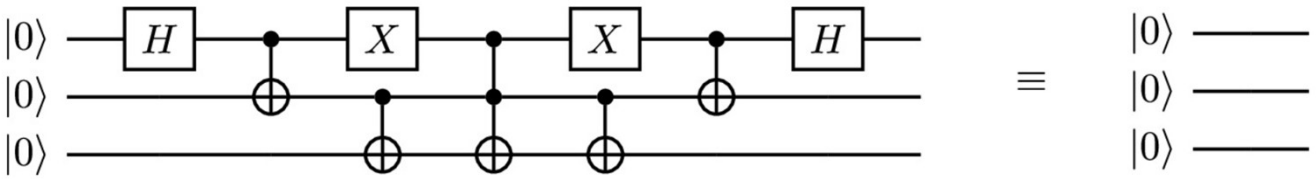
```
...
cx  q0, q2;
...
```

Classical vs. quantum compiler



Inspire



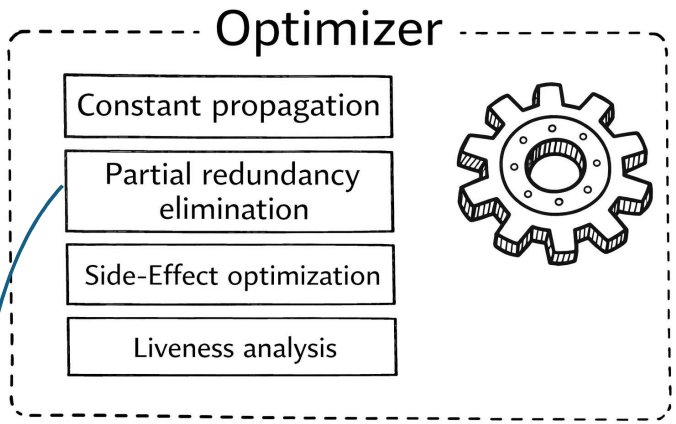



```
...
// whenever q0 in |1⟩, q1 is in |1⟩
ccx q0, q1, q2;
...
```

QCP

```
...
cx  q0, q2;
...
```

Classical vs. quantum compiler

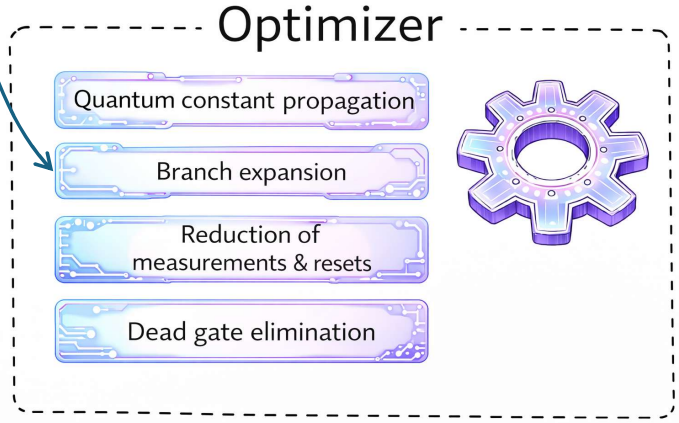



```
int t;
if (cond) {
   t = x + y;
   use(t);
}
int u = x + y;
use(u);
```

PRE

```
int t = x + y;
if (cond) {
   use(t);
}
use(t);
```

Inspire

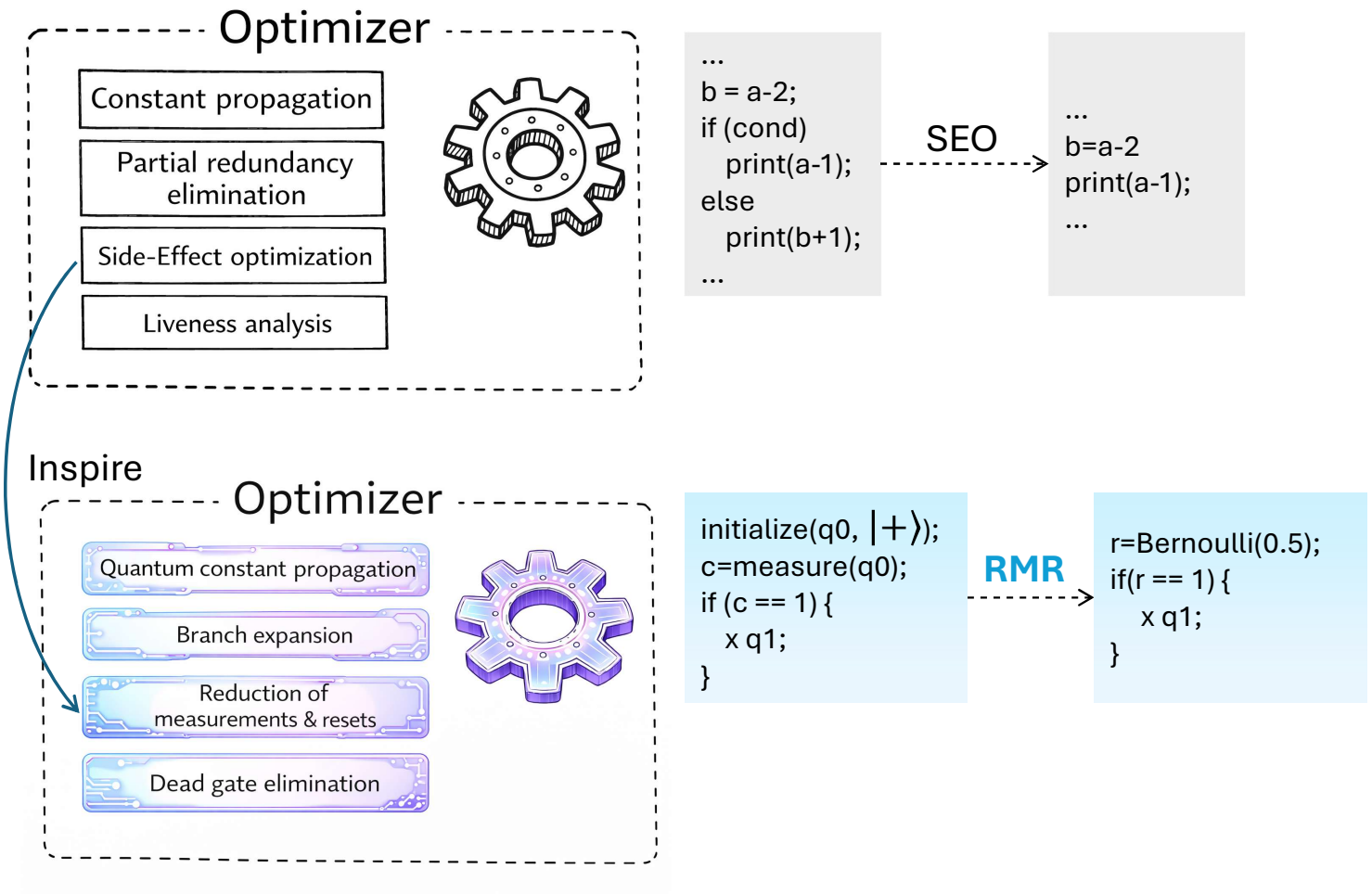



```
m = measure(q0);
x q1;
if(m) x q1;
else h q1;
h q1;
```

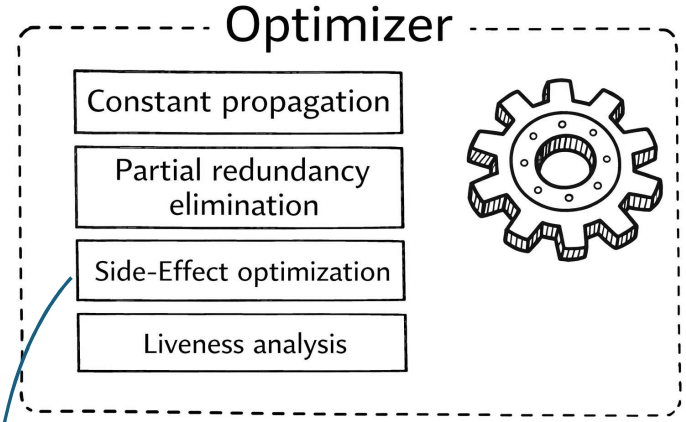
BE

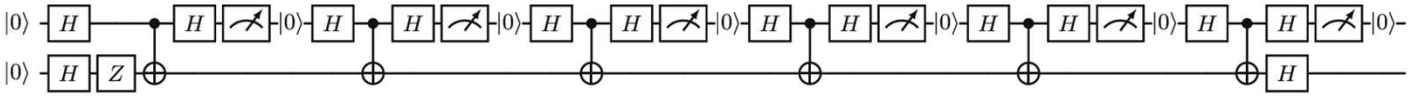
```
m = measure(q0);
if(m) h q1;
else x q1;
```

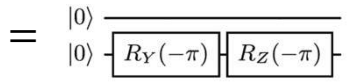
Classical vs. quantum compiler



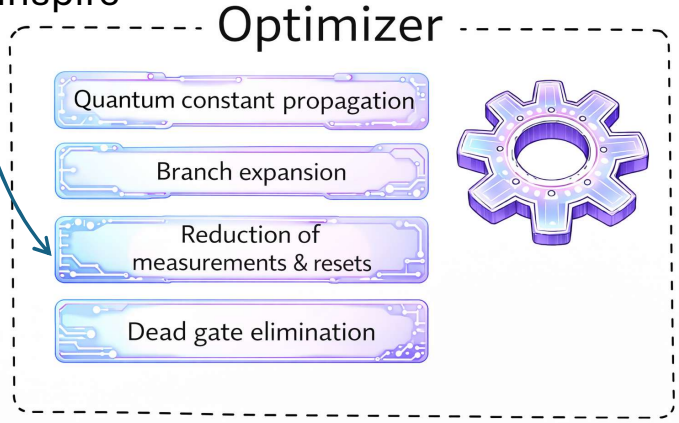
Classical vs. quantum compiler







Inspire

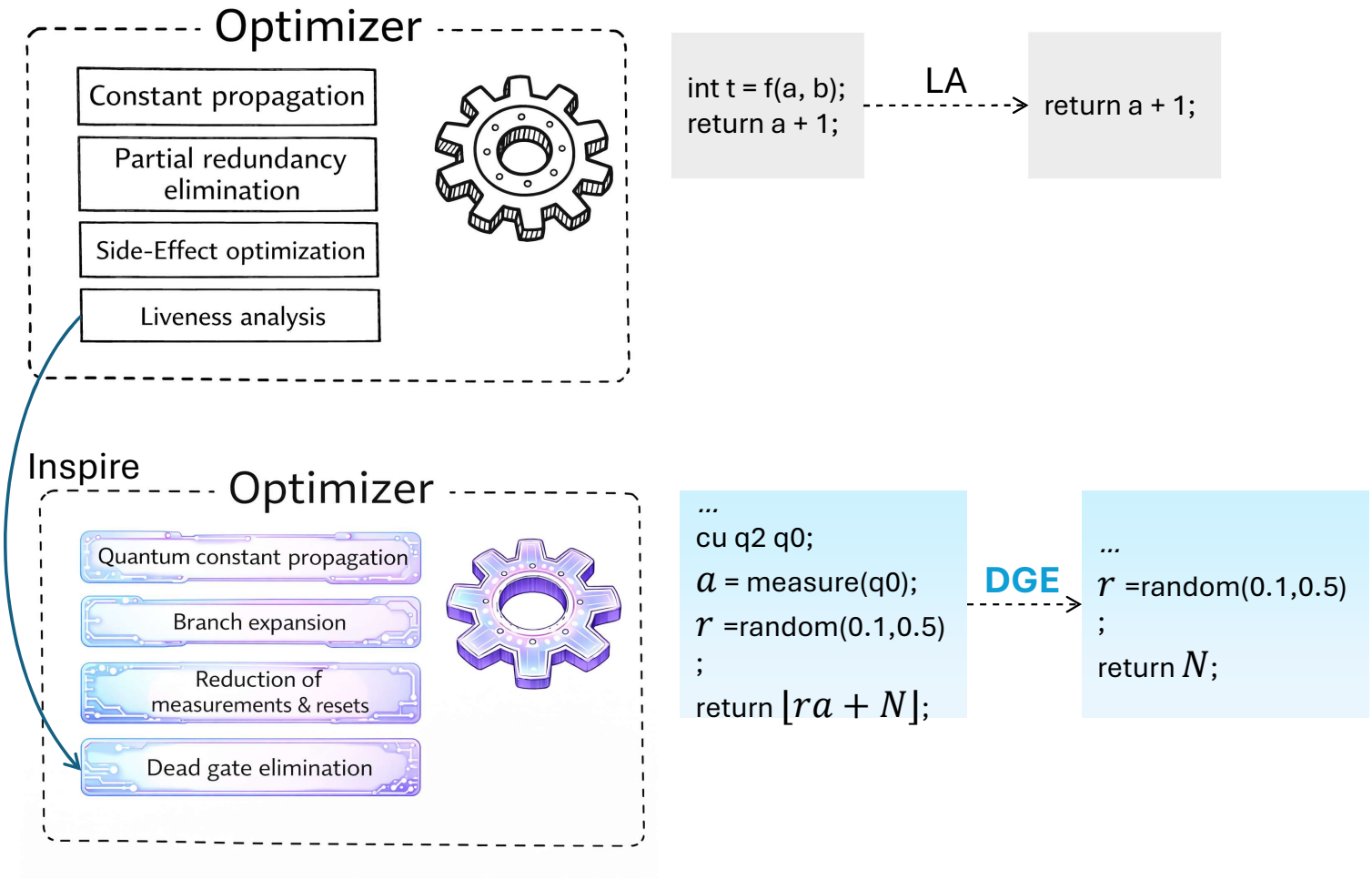



```
initialize(q0, |+⟩);
c=measure(q0);
if (c == 1) {
   x q1;
}
```

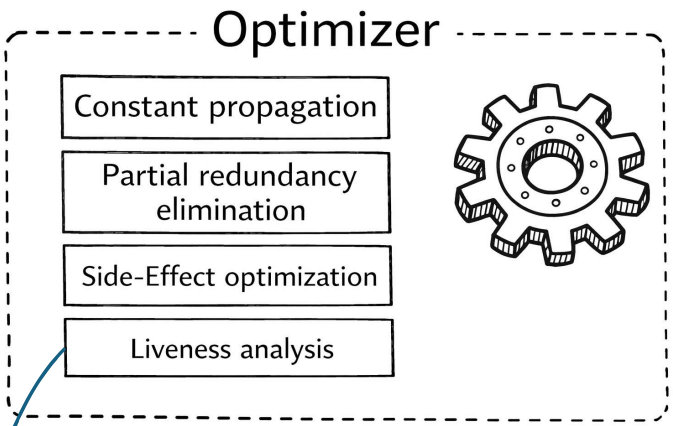
RMR

```
r=Bernoulli(0.5);
if(r == 1) {
   x q1;
}
```

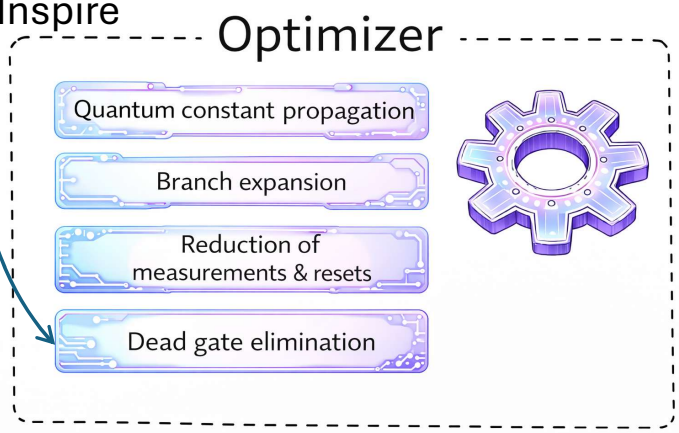
Classical vs. quantum compiler

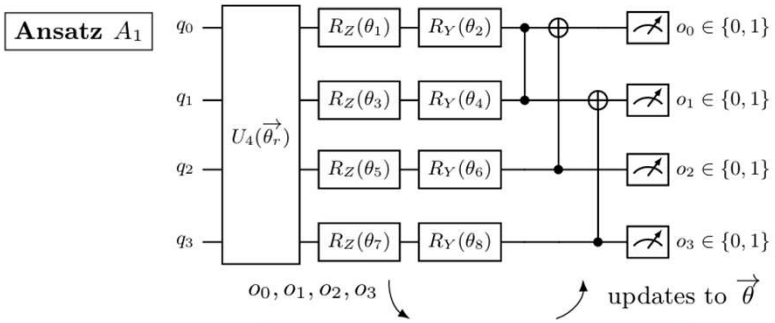


Classical vs. quantum compiler



Inspire



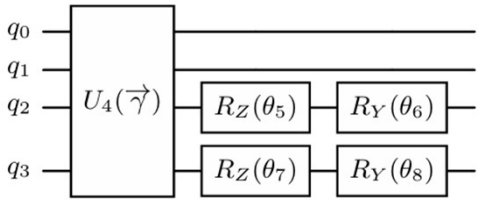


Optimizer

$$\mathbb{E}_Z \leftarrow \left\langle \psi(\vec{\theta}) \middle| Z_3 Z_4 \middle| \psi(\vec{\theta}) \right\rangle$$

$$\mathbb{E}_X \leftarrow \left\langle \psi(\vec{\theta}) \middle| X_3 X_4 \middle| \psi(\vec{\theta}) \right\rangle$$

updates $\leftarrow$ **combine** $(\mathbb{E}_Z, \mathbb{E}_X)$
return updates

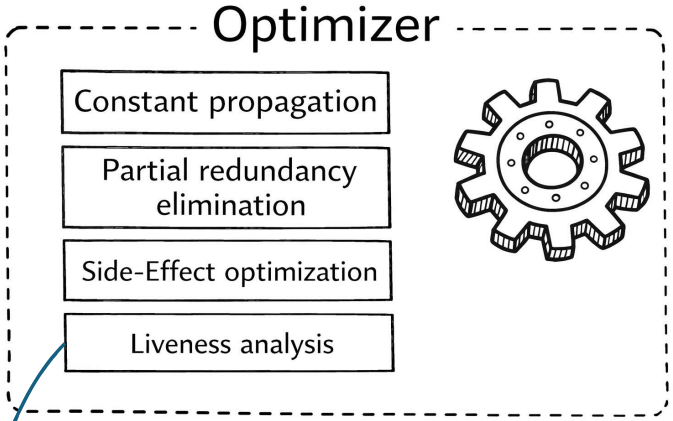


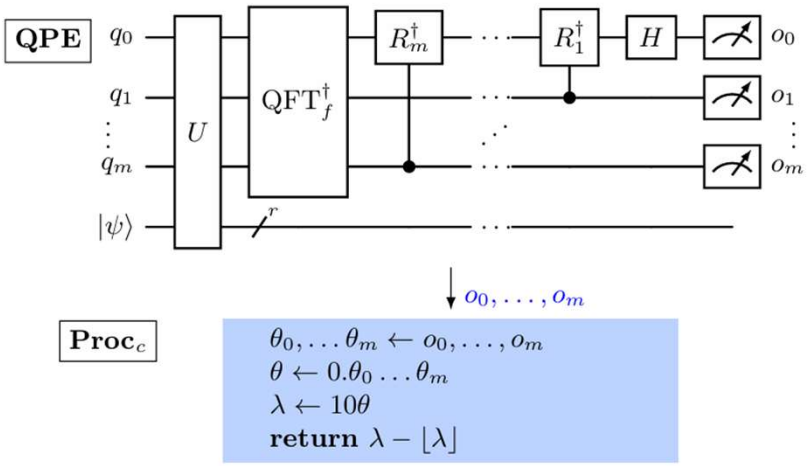

```
...
cu q2 q0;
a = measure(q0);
r =random(0.1,0.5)
;
return ⌊ra + N⌋;
```

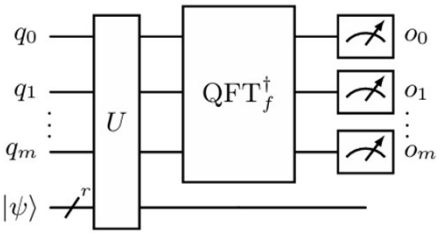
DGE

```
...
r =random(0.1,0.5)
;
return N;
```

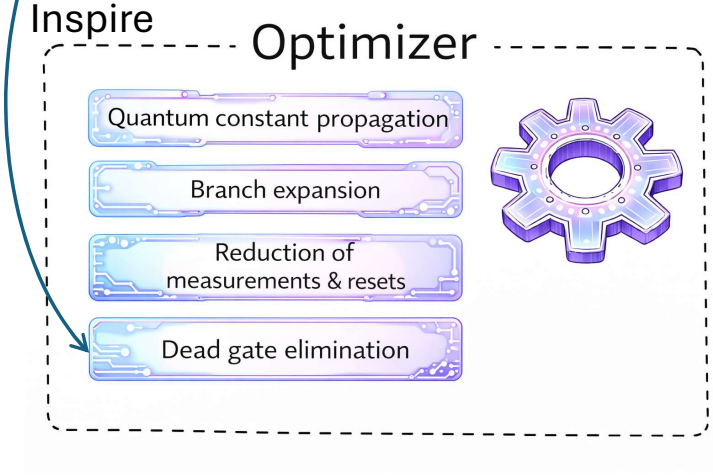
Classical vs. quantum compiler







Inspire

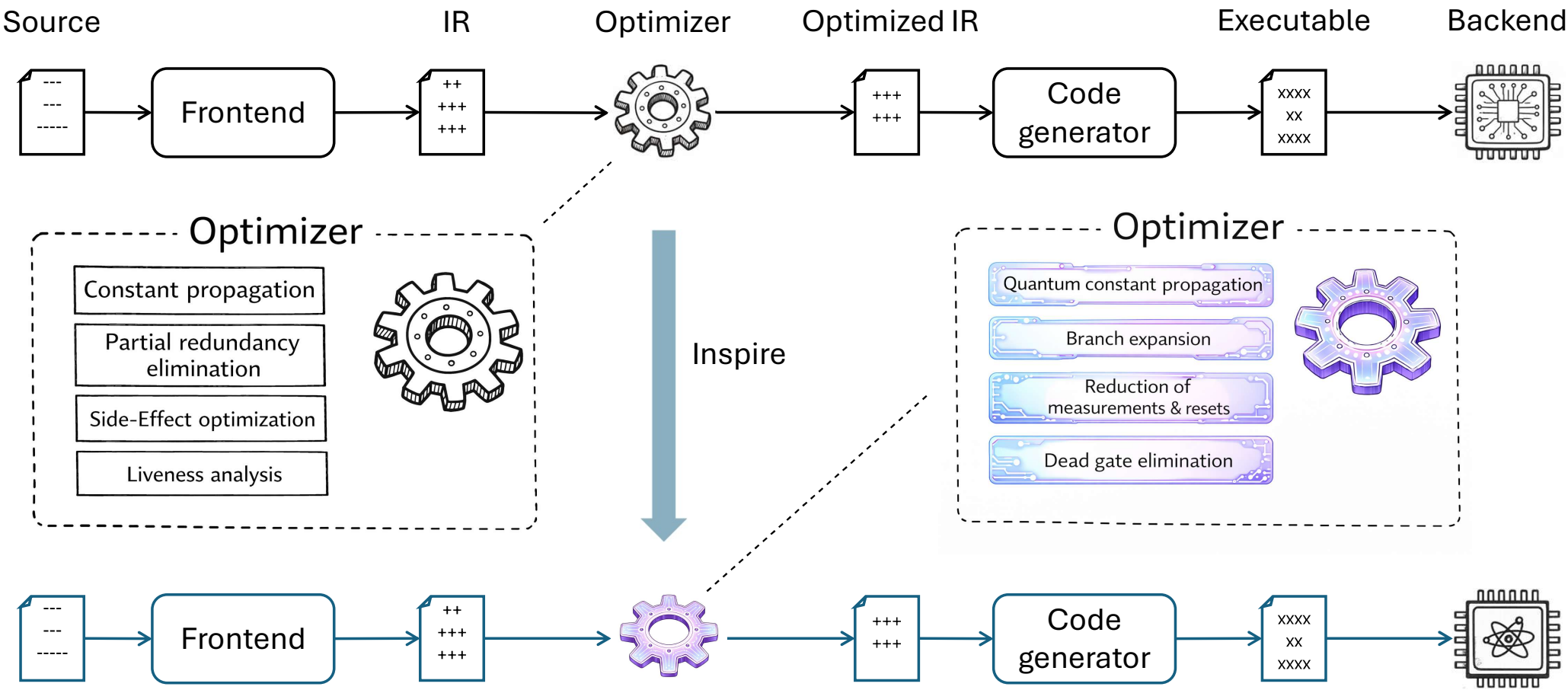



```
...
cu q2 q0;
a = measure(q0);
r =random(0.1,0.5)
;
return ⌊ra + N⌋;
```

DGE

```
...
r =random(0.1,0.5)
;
return N;
```

Classical vs. quantum compiler



The end

- Thanks for your attention!

References

[CFM25] Chen, Y.; Fulginiti, I.; Mendl, C. B.: Optimization Framework for Reducing Mid-circuit Measurements and Resets. In (Paszynski, M.; Barnard, A. S.; Zhang, Y. J., eds.): Computational Science – ICCS 2025 Workshops. Springer Nature Switzerland, Cham, pp. 150–164, 2025.

[Ch25] Chen, Y.: Unleashing Optimization in Dynamic Circuits through Branch Expansion. In: Proceedings of the 22nd ACM International Conference on Computing Frontiers. CF '25, Association for Computing Machinery, pp. 29–37, 2025, https://doi.org/10.1145/3719276.3725187.

[CMS25] Chen, Y.; Mendl, C. B.; Seidl, H.: Dead Gate Elimination. In (Lees, M. H. et al., eds.): Computational Science – ICCS 2025. Springer Nature Switzerland, Cham, pp. 135–150, 2025.

[CS23] Chen, Y.; Stade, Y.: Quantum Constant Propagation. In (Hermenegildo, M. V.; Morales, J. F., eds.): Static Analysis. Springer Nature Switzerland, Cham, pp. 164–189, 2023.