Pro demonstraci problému konceptu použijme následující kód:

Class A {

@ctime Bool staticBool = true;

}

Class B {

@ctime Bool staticBool = false;

}

Class C {

@ctime Int staticBool = 3;

}

@ctime {

Type[] typeArray = [A, B, C, A, B, C];

Int! i = 0;

Bool b = typeArray[i++].staticBool && typeArray[i++].staticBool;

}

Klíčovým je pro nás výraz na posledním řádku kódu. Je zřejmé, že díky tomu, že se pracuje s typovým polem, se výraz musí vykonávat v @ctime.

Kdyby se kód vykonával za běhu, sestavení kódu by probíhalo následujícím způsobem:

1. Odvodily by se typy pro obě strany logického výrazu (Bool a Bool)
2. Ve jmenném prostoru typu Bool by se provedla rezoluce pro volání #opBinary(Operator.binAnd, Bool arg)
3. Sestaví se sémantický strom pro volání dané funkce. Tato funkce je definovaná kompilátorem a využívá zkratového vyhodnocování (pokud je první argument false, druhý se ani nevyhodnocuje).

Tento přístup však nelze zcela využít pro @ctime kód: pro zjištění typu výrazu „typeArray[i++].staticBool“ totiž musíme výraz vyhodnotit. Díky tomu jsme oba výrazy vyhodnotili ještě před tím, než jsme se vůbec dostali k operátoru &&, který zkratové vyhodnocení používá.