CS-A1120 Ohjelmointi 2

Aalto-yliopisto
Perustieteiden
korkeakoulu

Luento 3
kevät 2026

Sanna Suoranta (suomeksi, AS2) ja
Lukas Ahrenberg (englanniksi, T1)

9.3.2026

https://presemo.aalto.fi/o2fi2026

Kertaus Boolen-operaatioista

- **Operaattorit olivat AND, OR, NOT ja XOR (Scalassa: &, |, !, ^).**
- **Alla on totuustaulut (syöte a, b) näille operaattoreille. Mikä merkinnöistä 🦍 🐳 🦑 🦆 vastaa mitä operaattoria?**
  - https://presemo.aalto.fi/o2fi2026

a	b	a 🦍 b
0	0	0
0	1	1
1	0	1
1	1	1

a	🐳a
0	1
1	0

a	b	a 🦑 b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a 🦆 b
0	0	0
0	1	0
1	0	0
1	1	1

Tänään O2:ssa

https://presemo.aalto.fi/o2fi2026

Kombinaatiologiikka

Päivän tavoite: ”kone” joka osaa laskea yhteen numeroita

Aalto-yliopisto
Perustieteiden
korkeakoulu

Kierroksen 3 oppimistavoitteet

- **Tämän viikon jälkeen,**
  - Pystyt tunnistamaan peruslogiikkaportit (logic gates) sekä lukemaan ja piirtämään yksinkertaisia piirikaavioita (circuit diagram)
  - Pystyt yhdistämään portteja ja väyliä (bus) rakentaaksesi loogisia funktioita (logic function)
  - Olet tietoinen kuinka Scalan rakenteita voi käyttää simuloimaan loogisia lausekkeita (logic expressions)

Kertaus viime viikolta

- **Bittiä käytetään ilmaisemaan kahta tilaa (0 / 1)**
- **Bittisekvenssiä voidaan käyttää ilmaisemaan tietoa (esim. numeroa)**
- **Boolen-operaatioita AND (ja), OR (tai), NOT (ei), … käytetään muuttamaan bittejä ja tietoa**

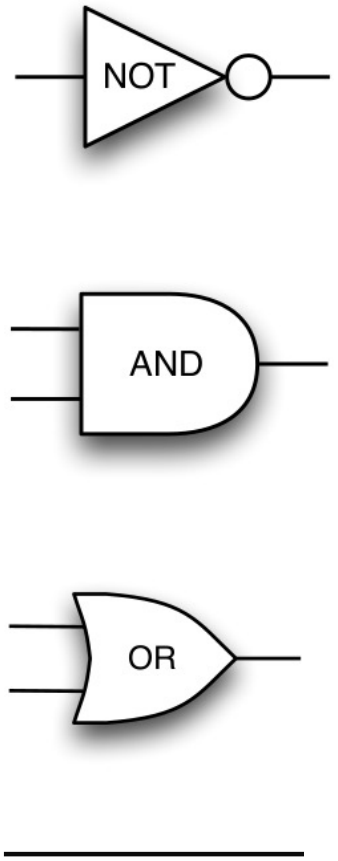
Millä periaatteilla tietokoneet toimivat?

- **Miten voimme rakentaa koneen, joka laskee bittien avulla?**
- **Laskenta:**
  - Ylhäältä–alas-lähestymistapa: Turingin koneet
  - Alhaalta–ylös-lähestymistapa: Portit ja piirit

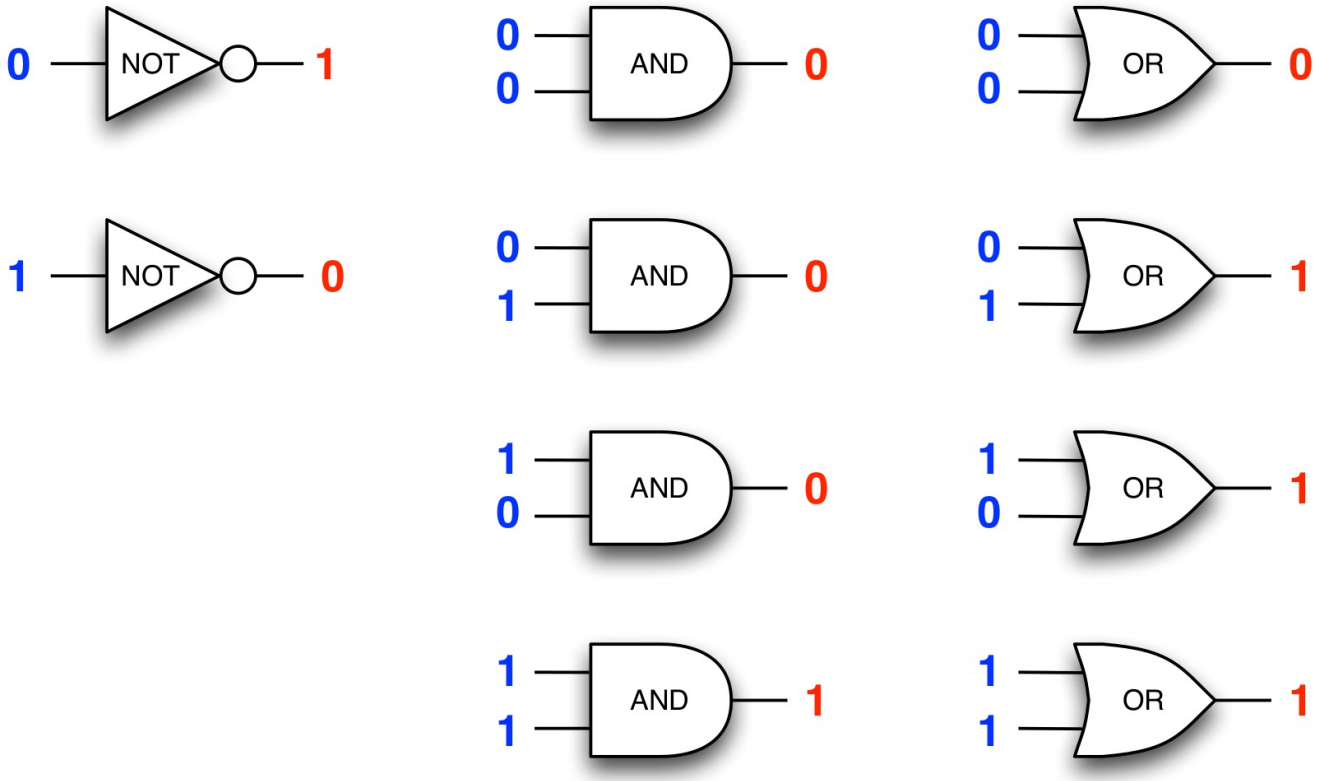
- **Turingin koneet käsitellään kurssilla CS-C2160 Laskennan teoria**

Rakennuspalikat tietokoneelle

- **Loogiset portit NOT, AND ja OR**
  - Esittävät Boolen-operaatioita
- **‘Johtimia’ tai ‘piuhoja’ (wire) porttien yhdistämiseksi**
- **Näiden avulla kurssin edetessä pystymme rakentamaan monimutkaisempia komponentteja**



Portit Boolen-operaattoreina



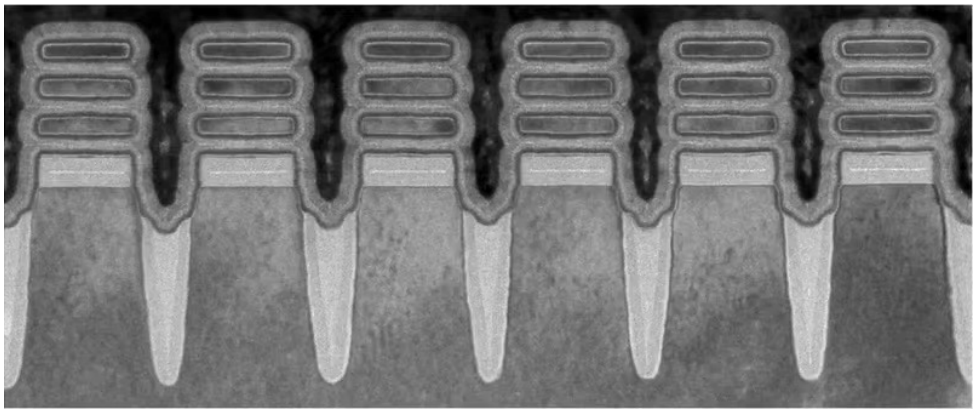
- **Portin ulostulon (output) arvo määräytyy syötteiden (input) perusteella**

Loogisen portin fyysiset toteutukset

- **Tyhjiöputkia (vacuum tube)**
- **Colossus (1944)**

https://www.tnmoc.org/colossus

- **2 nm – 5 nm transistorit**
- **Nykypäivää**

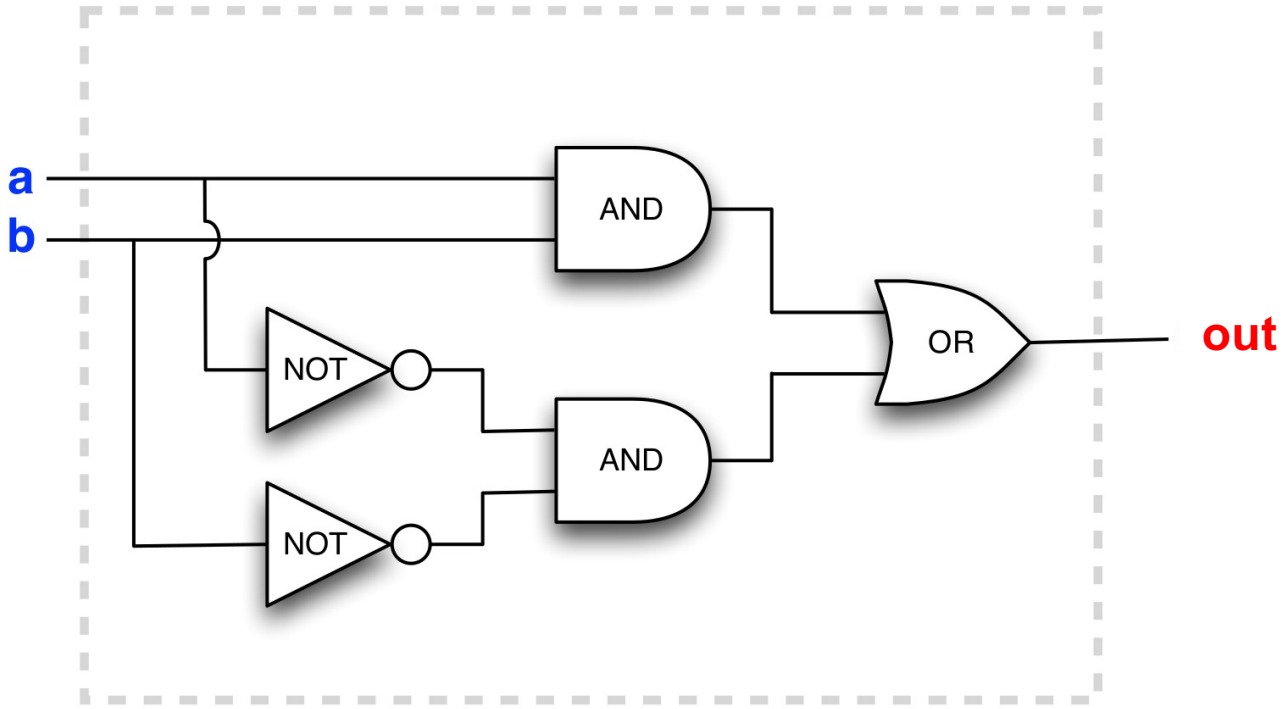


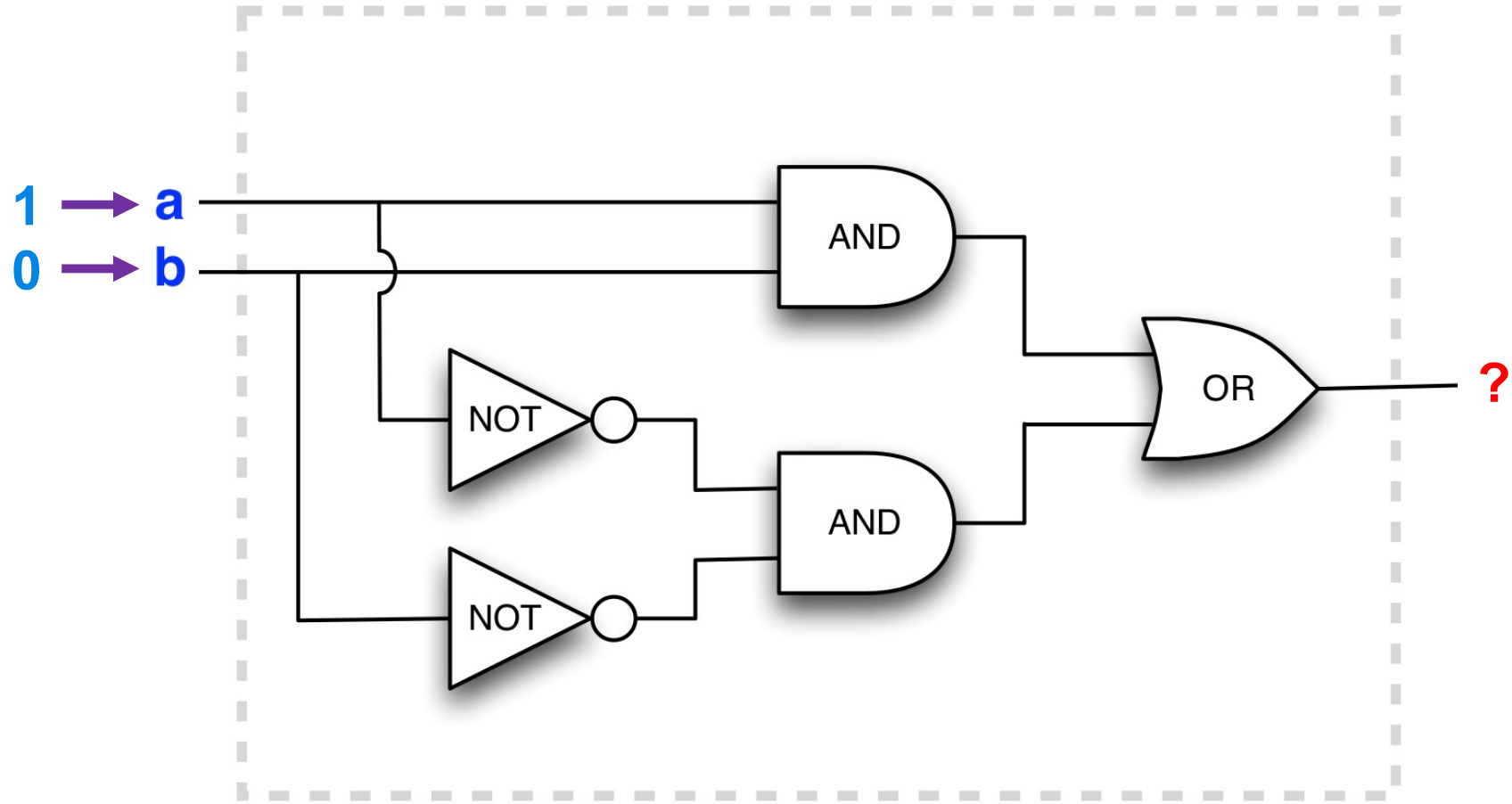
https://newatlas.com/computers/ibm-2-nm-chips-transistors/

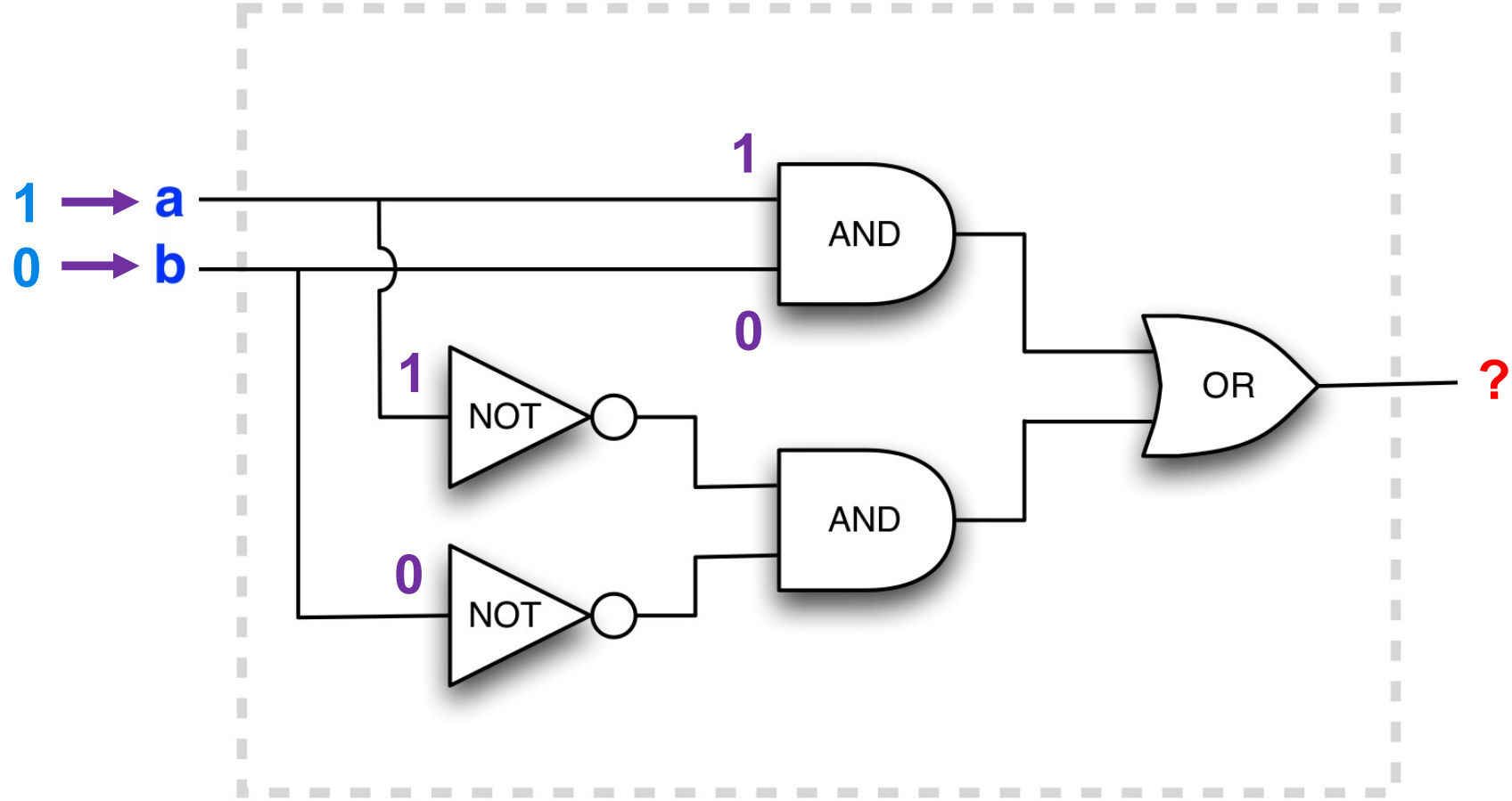
Millainen kone on mahdollista rakentaa yhdistämällä (combining) loogisia portteja?

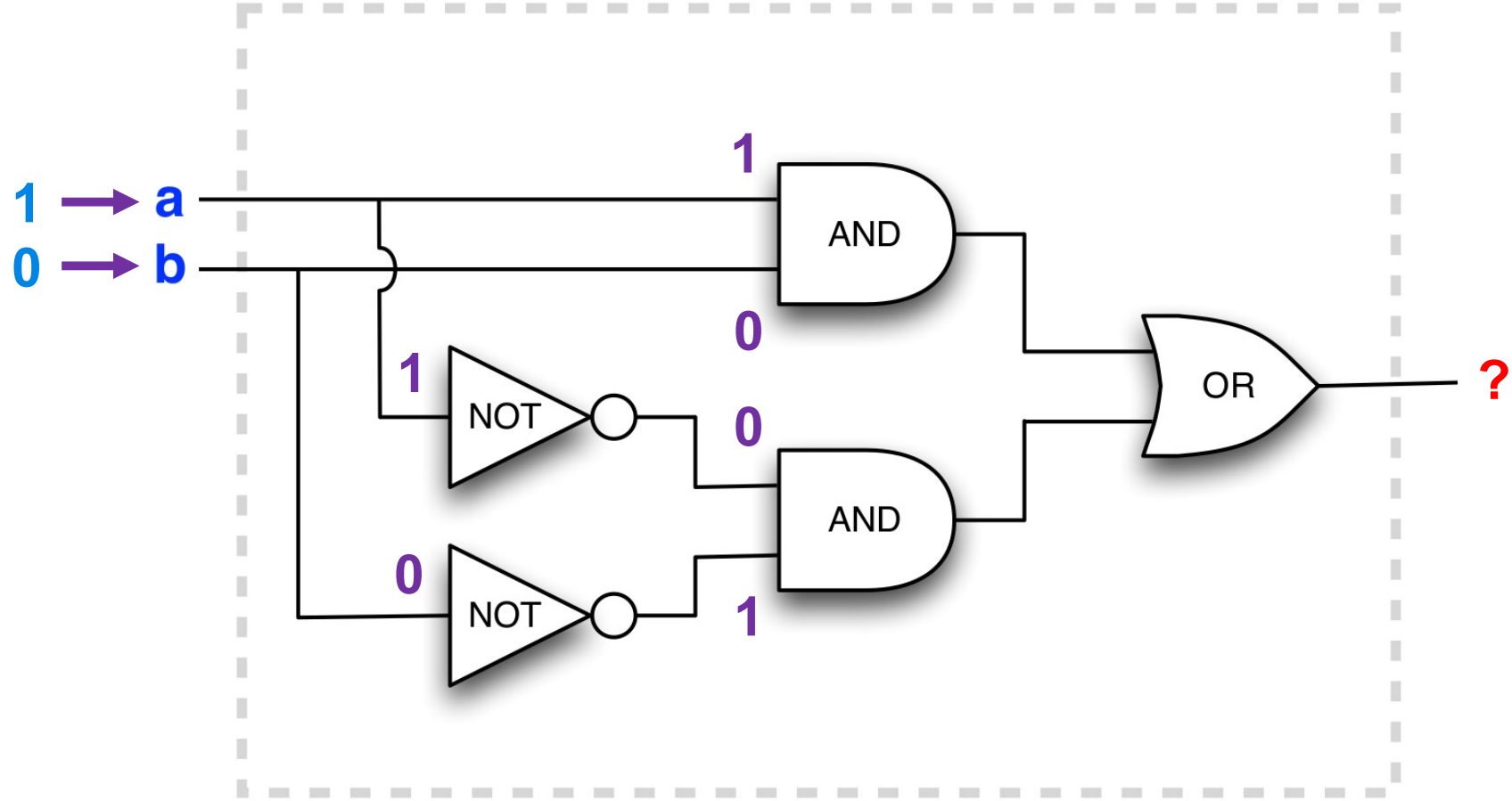
Piirit

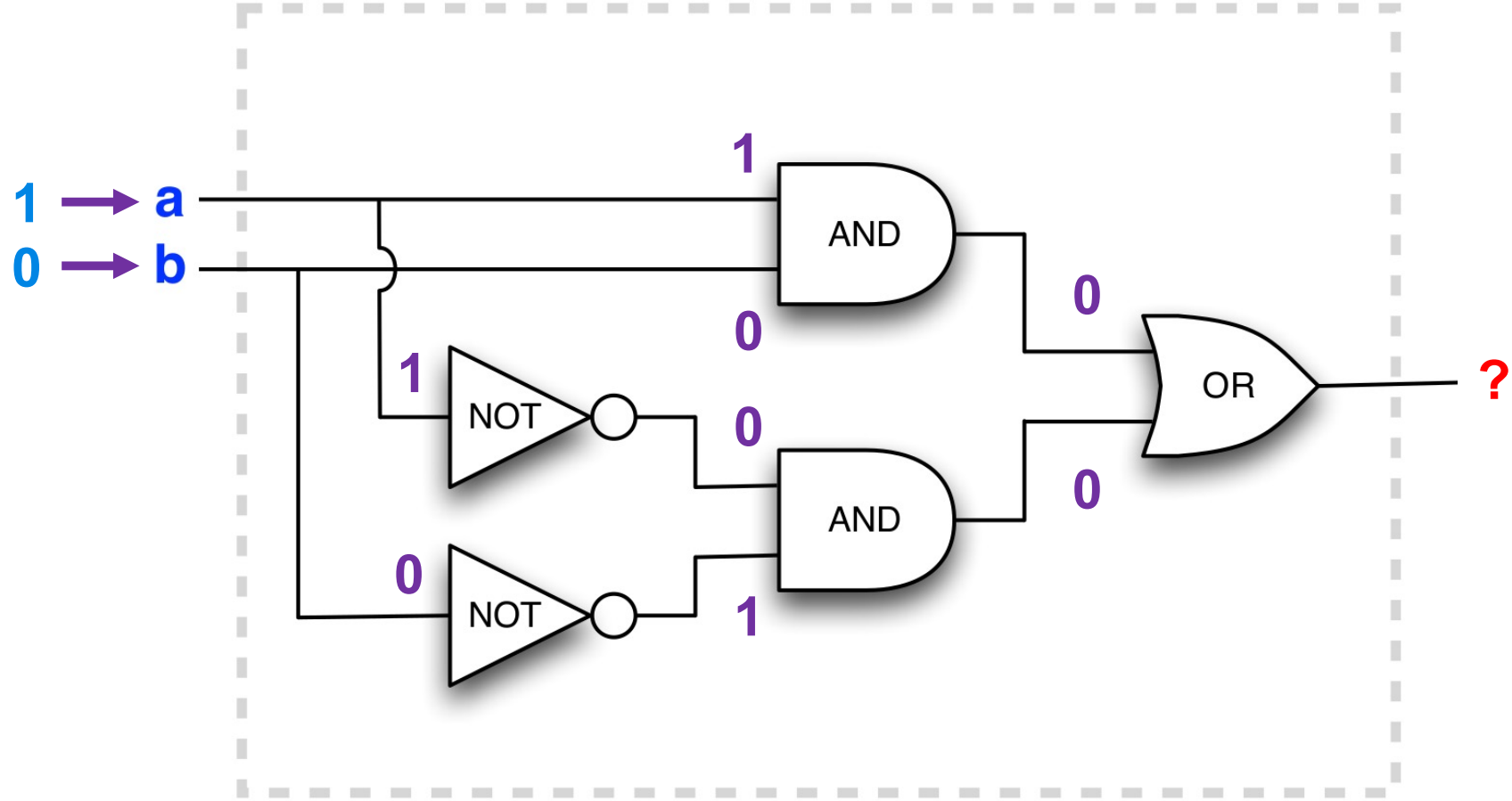
- Porttien yhdistelmää (combination) kutsutaan piiriksi (circuit)
- Piirillä on syötteitä (inputs) ja ulostulo (output), vrt. laatikko:

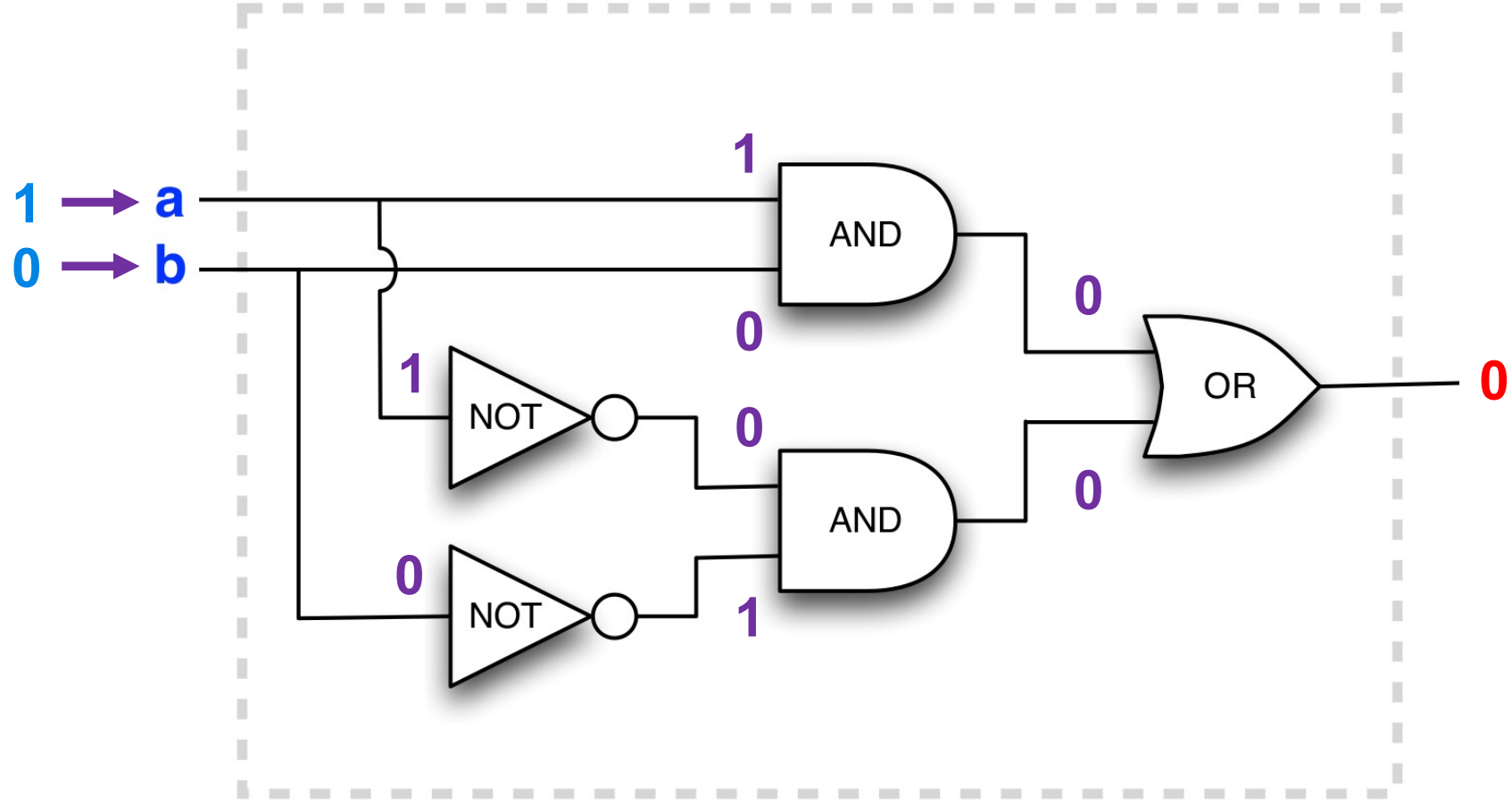



1
a
0
b
AND
NOT
AND
NOT
OR
?

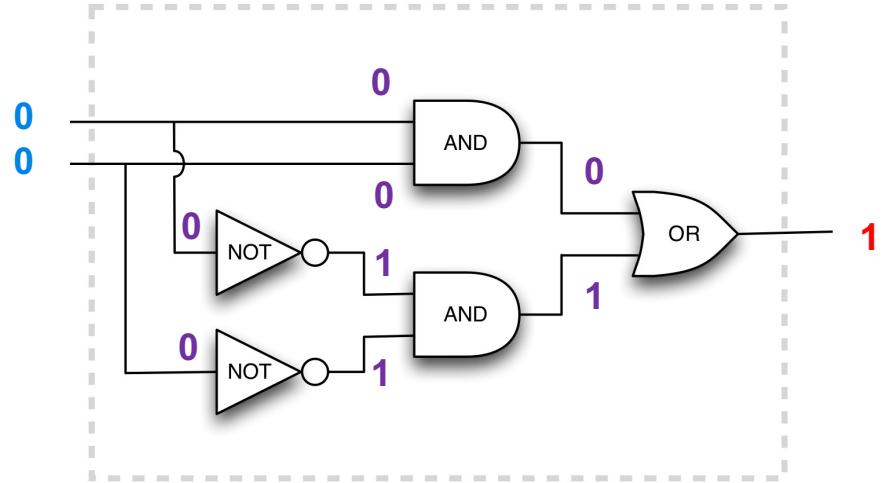

1
a
0
b
1
AND
0
1
NOT
AND
0
NOT
OR
?

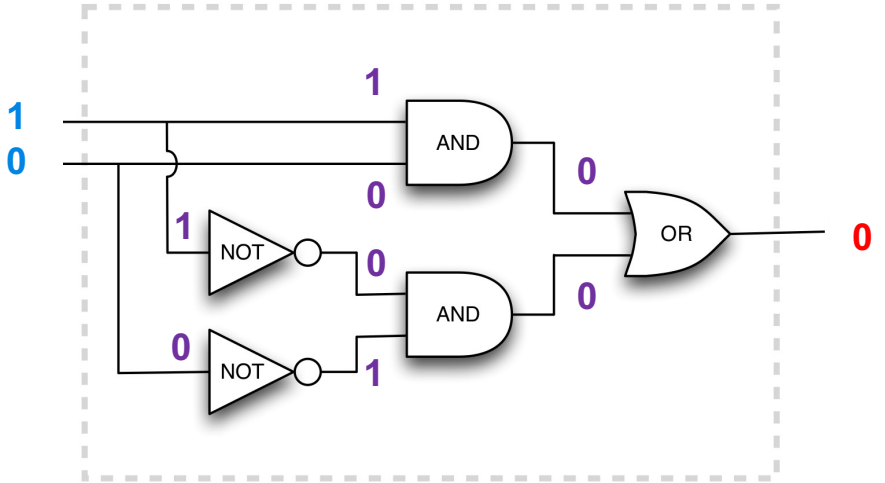

1
a
0
b
1
AND
0
1
NOT
0
0
AND
0
NOT
1
OR
?

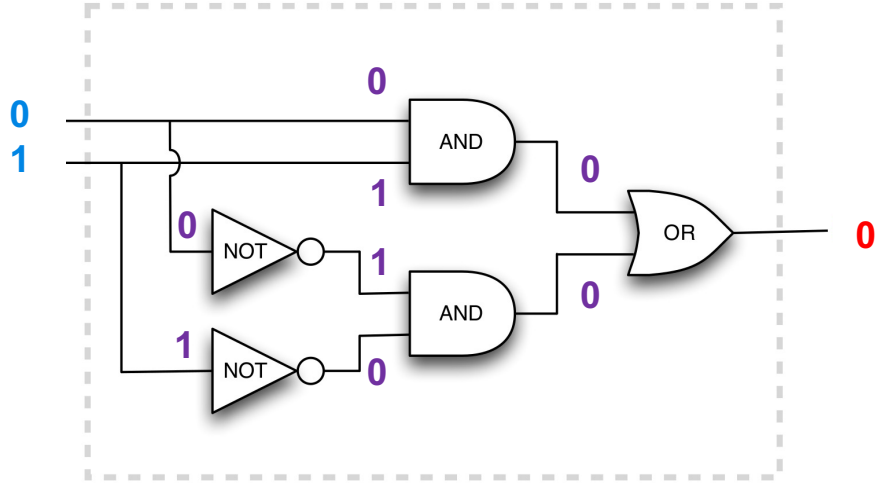

1
a
0
b
1
AND
0
1
NOT
0
0
AND
0
0
NOT
1
0
OR
0
?

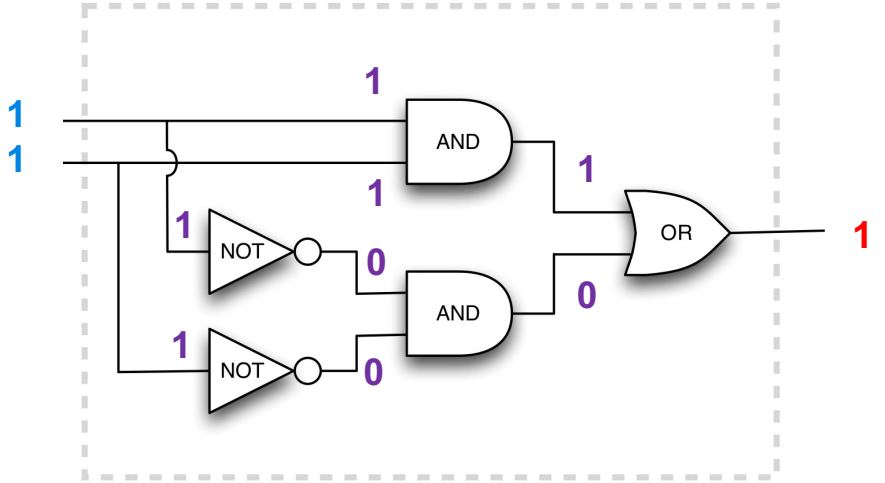

1
a
0
b
1
AND
0
0
1
NOT
0
0
AND
0
0
NOT
1
OR
0

Piirin ulostulo muuttuu syötteiden perusteella!





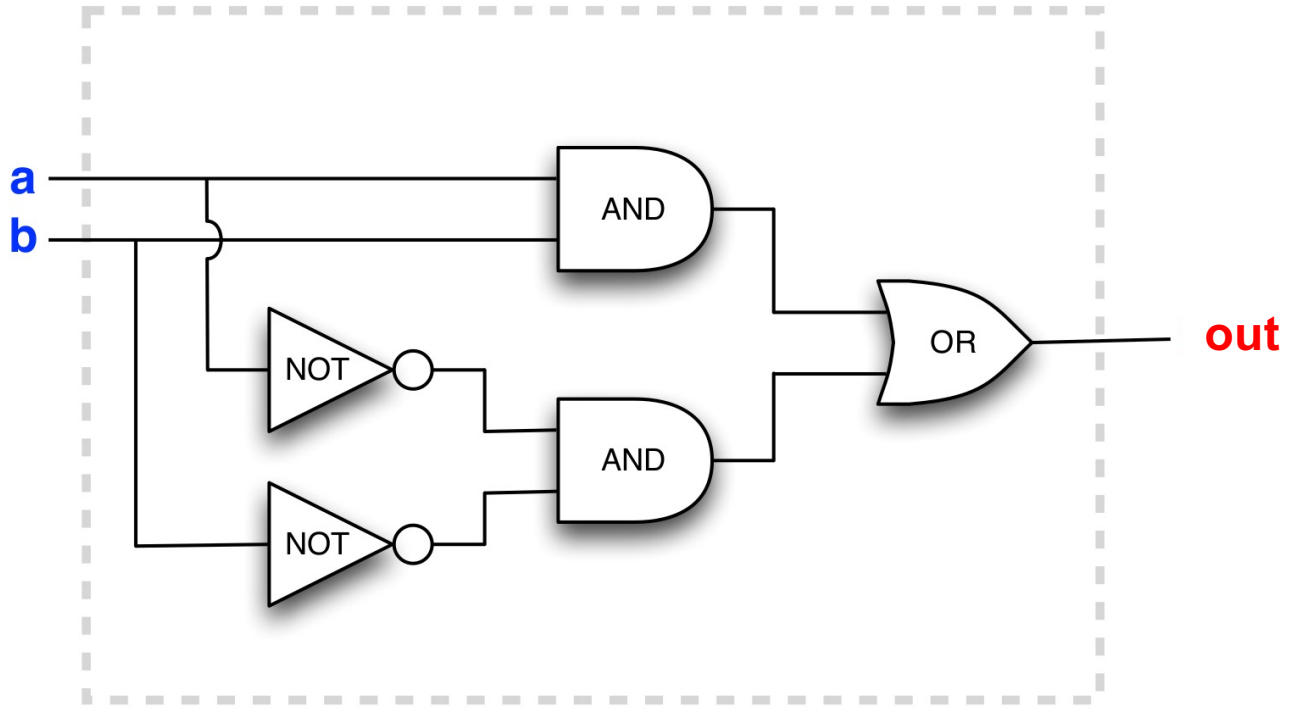




Mitä tämä piiri oikeastaan laskee?

a) Mitä Boolen-lauseketta tämä piiri vastaa?

b) Minkä ‘funktion’ se laskee, ilmaistuna omin sanoin yhdellä virkkeellä?



Totuustaulu (truth table)

a	b	(a & b) | (!a & !b)
0	0	1
0	1	0
1	0	0
1	1	1

Kierrokselta 1 – lausekkeita esittävät luokat

- **Muistathan nämä luokat kierrokselta 1?**
  - Numeerisille arvoille:

```
abstract class Value:
  def read: Int
end Value

class VariableValue(c: Int) extends Value:
  protected var _myVal = c
  def write(v: Int) =
    _myVal = v
  def read = _myVal
end VariableValue
```

  - Summalle

```
class SumValue(v1: Value, v2: Value)
extends Value:
  def read = v1.read + v2.read
end SumValue
```

- **Näillä voimme luoda muuttujia ja summia, esim.**

```
val v1 = VariableValue(10)
val v2 = VariableValue(20)
val s = SumValue(v1,v2)

println(f"The sum is now ${s.read}")

v1.write(0)

println(f"The sum is now ${s.read}")
```

- **Tämä tulostaa**

```
The sum is now 30
The sum is now 20
```

- **Kokeile scastie:ssa:**
  https://scastie.scala-lang.org/UYb3hWMVSiaBDb576JCLIA

Kierrokselta 1 – lausekkeita esittävät luokat

- Numeerisille arvoille:

```
abstract class Value:
  def read: Int
end Value

class VariableValue(c: Int) extends
Value:
  protected var _myVal = c
  def write(v: Int) =
    _myVal = v
  def read = _myVal
end VariableValue
```

- Summalle

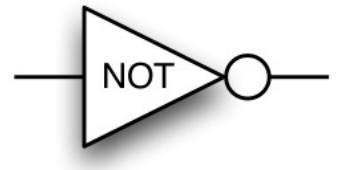
```
class SumValue(v1: Value, v2: Value)
extends Value:
  def read = v1.read + v2.read
end SumValue
```

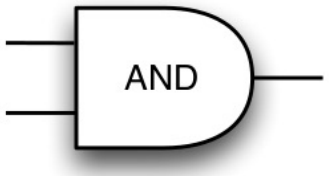
- **Ilman että ajat koodin, mitä tämä tulostaa?**

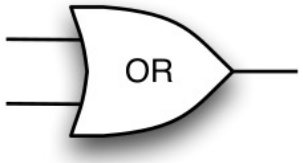
```
val x = VariableValue(10)
val y = VariableValue(20)
val z = VariableValue(30)

val sum1 = SumValue(SumValue(x,y),z)
val sum2 = VariableValue((x.read+y.read) +z.read)

print(f"A:${sum1.read},")
print(f"B:${sum2.read},")

x.write(100)

print(f"C:${sum1.read},")
print(f"D:${sum2.read},")
```

Piirien jäljittely (emulate) Scalalla

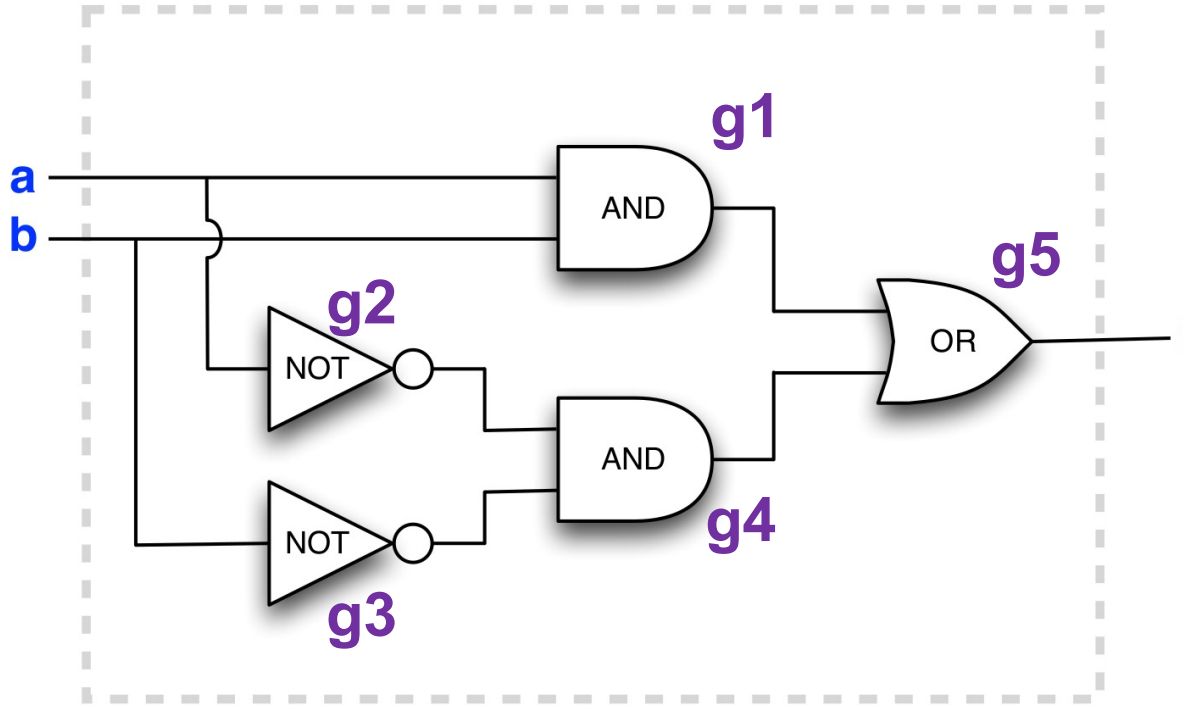
- **Voidaksemme jäljitellä (emuloida) piirien toimintaa (ja myöhemmin tietokonetta), kurssilla kirjoitetaan ohjelmoijan rakennussarja Scalalla**
- **Meillä pitää olla keinot esittää:**
  - NOT
  - AND
  - OR
  - piuhat



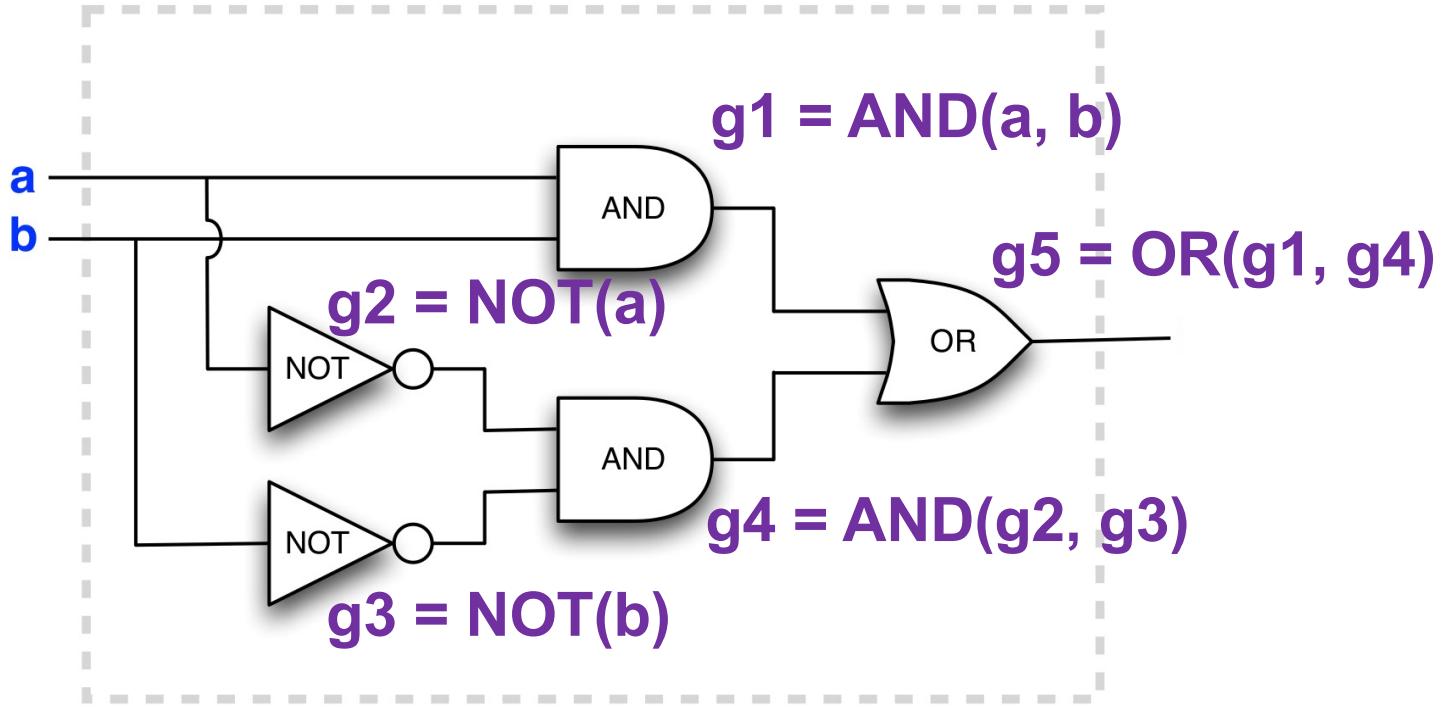




Käytetään olioita esittämään portteja ja syötteitä



Syöte on ‘edellisen’ portin ulostulo tai koko piirin syöte-elementti



- Tämän pitäisi hoitaa johdoitus
- Ulostulo on ketjun viimeisen olion arvo

`tinylog` –Scala-paketin teko

```
package tinylog
```

- **Käytetään yhteistä abstraktia luokkaa sekä piireille että syötteille**

```
abstract class Gate():
  def value: Boolean // implemented by the extending classes
end Gate
```

- **Portin arvo (value) on sen ulostulon arvo**
  - True (1) ja False (0)
  - Huom! Value on funktio!

Loogisten portien toteutus …

```
class NotGate(in: Gate) extends Gate():
  def value = !in.value
end NotGate

class OrGate(in1: Gate, in2: Gate) extends Gate():
  def value = in1.value || in2.value
end OrGate

class AndGate(in1: Gate, in2: Gate) extends Gate():
  def value = in1.value && in2.value
end AndGate
```

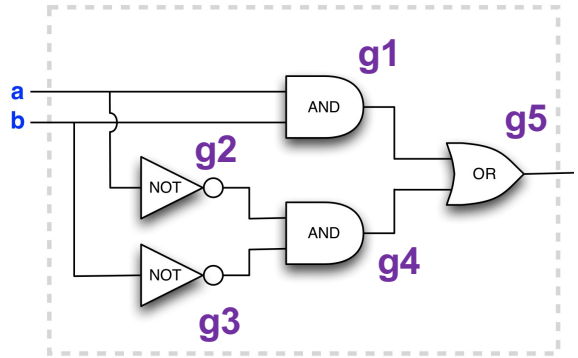
… ja syötteiden toteutus

```
class InputElement() extends Gate():
  var v = false // default value is false
  def set(s: Boolean) = { v = s }
  def value = v
end InputElement
```

- **Muistatko polynomi-tehtävän ensimmäiseltä kierrokselta?**

Esimerkkipiirin rakentaminen Scalalla

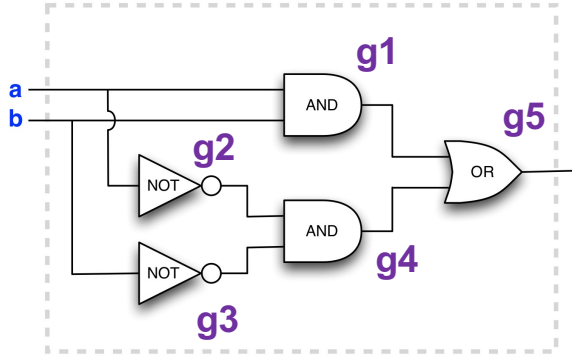
```
import tinylog.*
// Build a simple circuit
val a = new InputElement()
val b = new InputElement()
val g1 = new AndGate(a,b)
val g2 = new NotGate(a)
val g3 = new NotGate(b)
val g4 = new AndGate(g2,g3)
val g5 = new OrGate(g1,g4)
```



- **Koodi löytyy kierroksen 3 paketista**

Piirin testaaminen

```
scala> a.set(true)
scala> b.set(false)
scala> g5.value
val res1: Boolean = false

scala> a.set(false)
scala> b.set(false)
scala> g5.value
val res2: Boolean = true
```



- **Huomaa kuinka piiri toimii funktiona: kun muutamme syötteitä, se ‘laskee uudelleen’ ulostulon**
  - Piiriä itseään ei muuteta mihinkään

Syntaksin parantaminen määrittelemällä operaattorit Scalassa

- **Luokalle voi määritellä omat operaattorit (esim. && || + - / …)**
  - Lyhennysmerkinnöiksi nyt && AND-, || OR- ja ! NOT-porteille

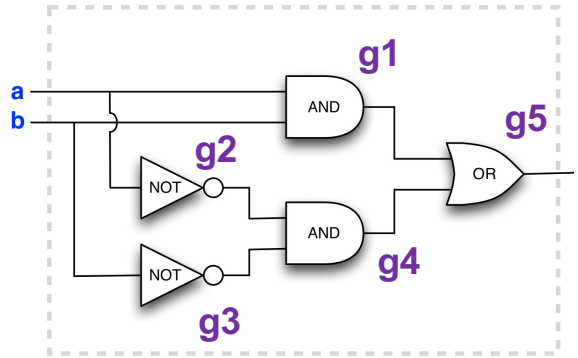
```
class Gate():
  /** Boolean value of the Gate.*/
  def value: Boolean

  /** Operator definition for NotGate of this Gate.*/
  def unary_! = new NotGate(this)
  /** Operator definition for AndGate of this Gate and that Gate.*/
  def &&(that: Gate): Gate = new AndGate(this, that)
  /** Operator definition for OrGate of this Gate and that Gate.*/
  def ||(that: Gate): Gate = new OrGate(this, that)
end Gate
```

Operaattoreiden käyttäminen

- **Nyt voimme tehdä näin:**

```
import tinylog.*
// Build a simple circuit
val a = InputElement()
val b = InputElement()
val g1 = a && b
val g2 = !a
val g3 = !b
val g4 = g2 && g3
val g5 = g1 || g4
```

- **Ohjelmointi on helpompaa ☺**



- **Suurienkin piirien tekeminen on helppoa, koska Scala luo objektit epäsuorasti:**

```
import tinylog.*
// Build a simple circuit
val a = InputElement()
val b = InputElement()
val g5 = (a && b) || (!a && !b)
```

Entä se kone, joka voi laskea yhteen numeroita ?

Aalto-yliopisto
Perustieteiden korkeakoulu

Aloitetaan kahdesta yksibittisestä numerosta

- **Binäärilukujen yhteenlasku**

```
0 + 0 =  0
0 + 1 =  1
1 + 0 =  1
1 + 1 = 10
```

- **Muista koulusta: kun summa ei ‘mahdu’ paikkaan, me lisäämme sen seuraavaan paikkaan muistinumeron (carry) avulla**

- **Kun muistinumero on kirjoitettu näkyviin:**

```
0 + 0 = 00
0 + 1 = 01
1 + 0 = 01
1 + 1 = 10
```

Nyt tiedämme syötteet ja lopputuloksen

- **Mutta millaisen piirin tarvitsemme?**
  - Eli mikä on lauseke (expression)?

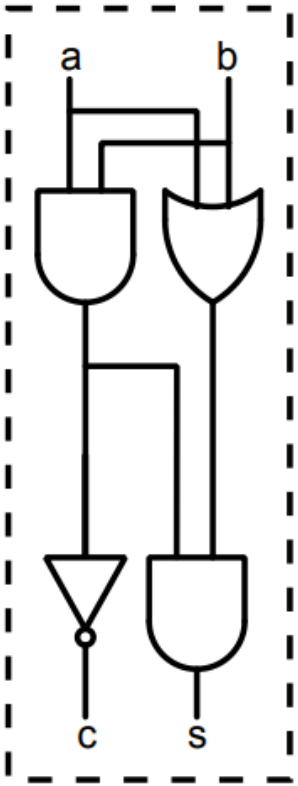
a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Muistin tueksi:

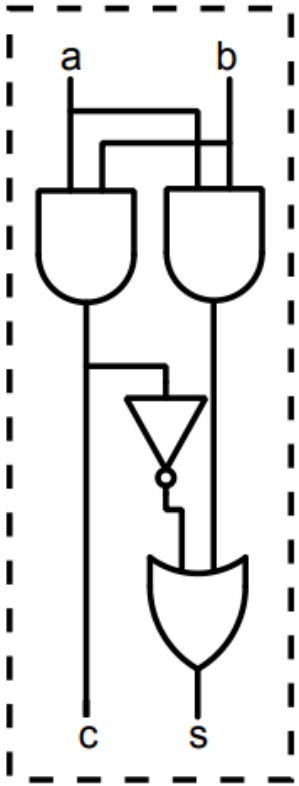
Mikä näistä piireistä tekee yhden bitin yhteenlaskun?

https://presemo.aalto.fi/o2fi2026

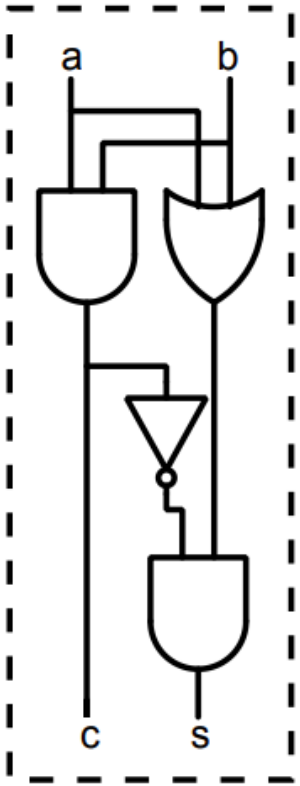
a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



1



2

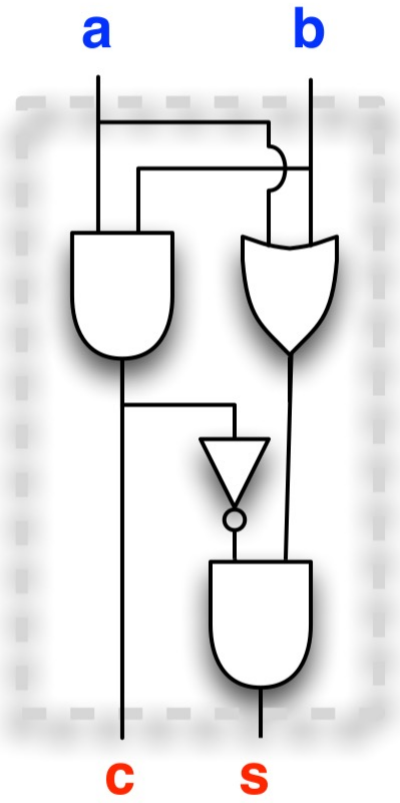


3

Muistin tueksi:

Tämä yhdistelmä?

?



Muistin tueksi:

Synteesi totuustaulu(i)sta

1. **Jokaiselle riville, jossa ulostulo on 1, kirjoita AND-operaatio, joka on totta tasan silloin, kun syötteillä on kyseisen rivin arvot**
2. **Yhdistä nämä ilmaisut yhdeksi pitkäksi OR-ilmaisuksi**

a	b	c	
0	0	0	
0	1	0	
1	0	0	
1	1	1	(a && b)

```
val c = (a && b)
```

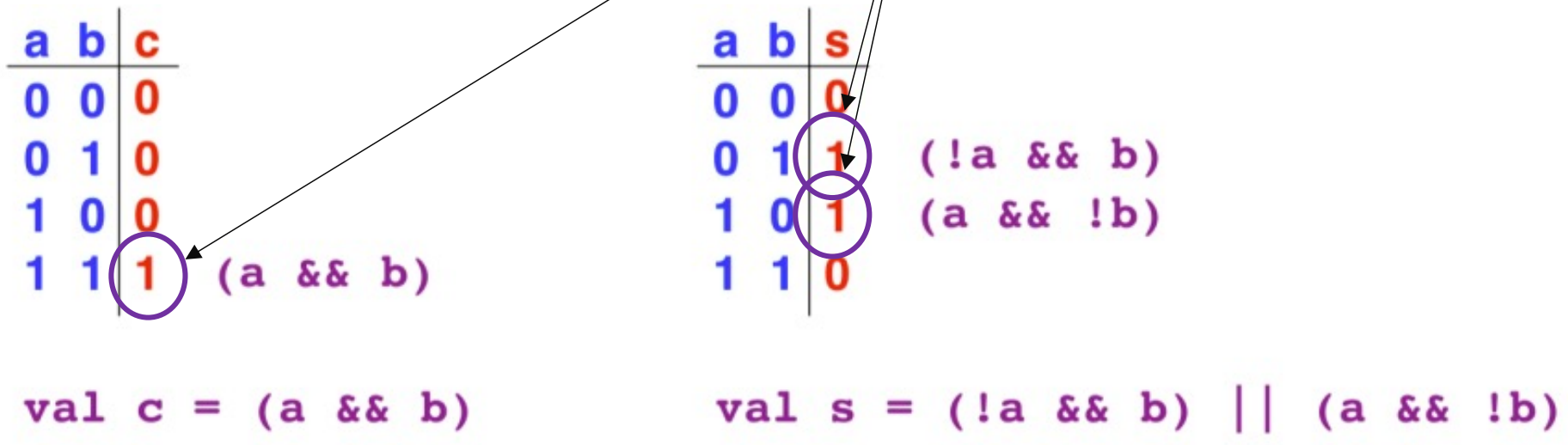
a	b	s	
0	0	0	
0	1	1	(!a && b)
1	0	1	(a && !b)
1	1	0	

```
val s = (!a && b) || (a && !b)
```

Olettaen että syöteet (a, b) on `tinylogGate`-olioita, Scala luo automaattisesti piirit näistä ilmiasuista!

Synteesi totuustaulu(i)sta

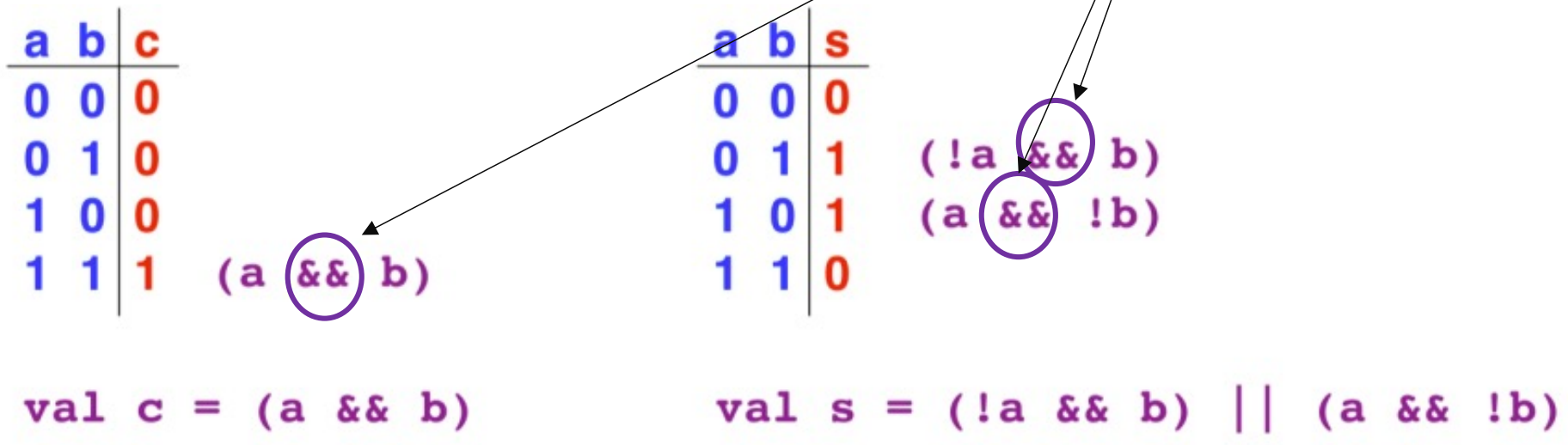
1. **Jokaiselle riville, jossa ulostulo on 1,** kirjoita AND-operaatio, joka on totta tasan silloin, kun tuloilla on kyseisen rivin arvot
2. Yhdistä nämä ilmaisut yhdeksi pitkäksi OR-ilmaisuksi



Olettaen että syöteet (a, b) on `tinylogGate`-olioita, Scala luo automaattisesti piirit näistä ilmiasuista!

Synteesi totuustaulu(i)sta

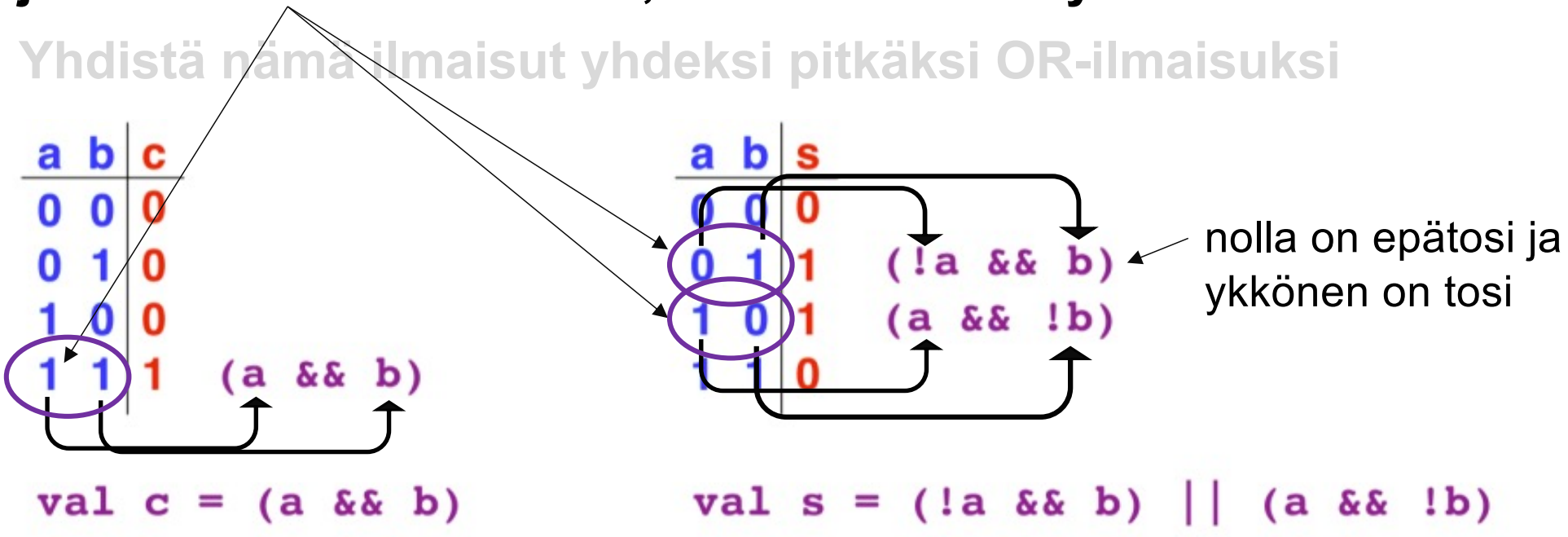
1. Jokaiselle riville, jossa ulostulo on 1, **kirjoita AND-operaatio,** joka on totta tasan silloin, kun tuloilla on kyseisen rivin arvot
2. Yhdistä nämä ilmaisut yhdeksi pitkäksi OR-ilmaisuksi



Olettaen että syöteet (a, b) on `tinylogGate`-olioita, Scala luo automaattisesti piirit näistä ilmiasuista!

Synteesi totuustaulu(i)sta

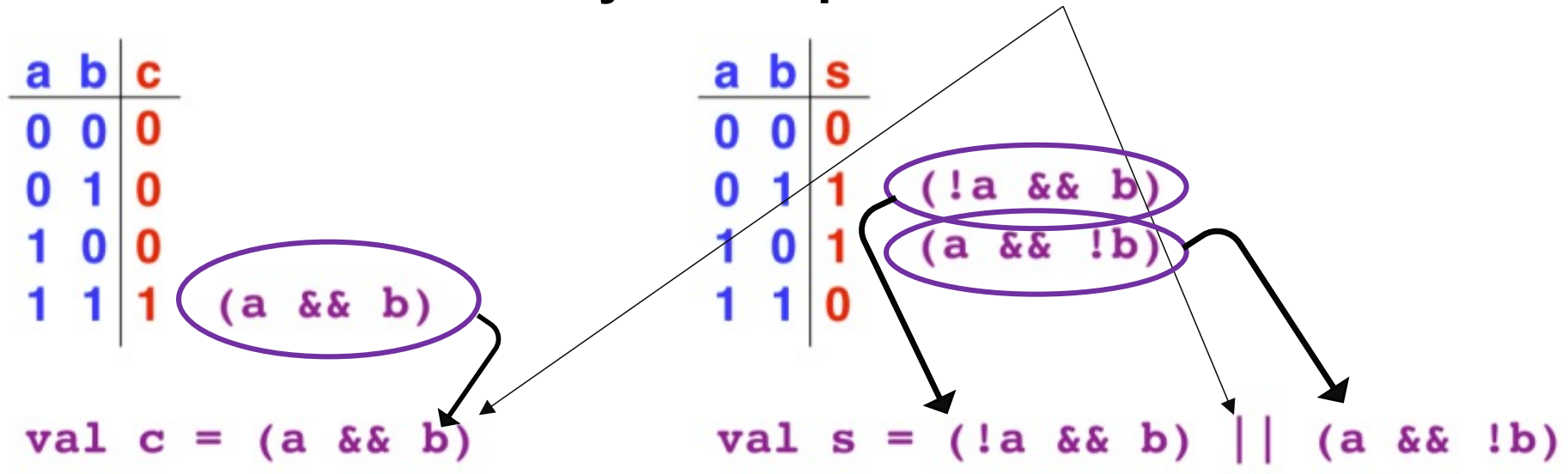
1. Jokaiselle riville, jossa ulostulo on 1, kirjoita AND-operaatio, **joka on totta tasan silloin, kun tuloilla on kyseisen rivin arvot**
2. Yhdistä nämä ilmaisut yhdeksi pitkäksi OR-ilmaisuksi



Olettaen että syöteet (a, b) on `tinylogGate`-olioita, Scala luo automaattisesti piirit näistä ilmiasuista!

Synteesi totuustaulu(i)sta

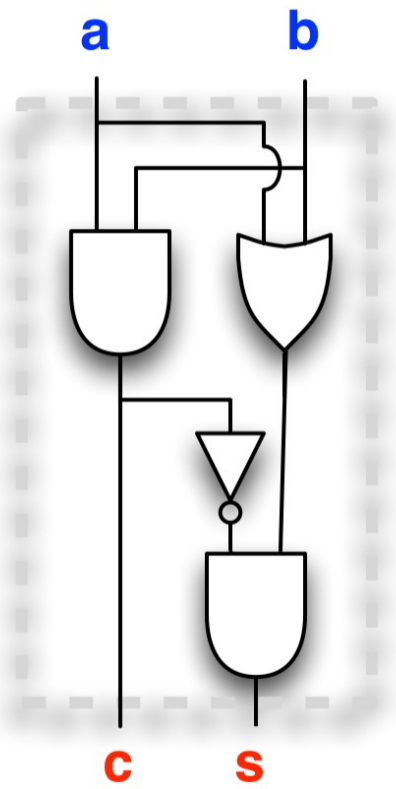
1. Jokaiselle riville, jossa ulostulo on 1, kirjoita AND-operaatio, joka on totta tasan silloin, kun tuloilla on kyseisen rivin arvot
2. **Yhdistä nämä ilmaisut yhdeksi pitkäksi OR-ilmaisuksi**

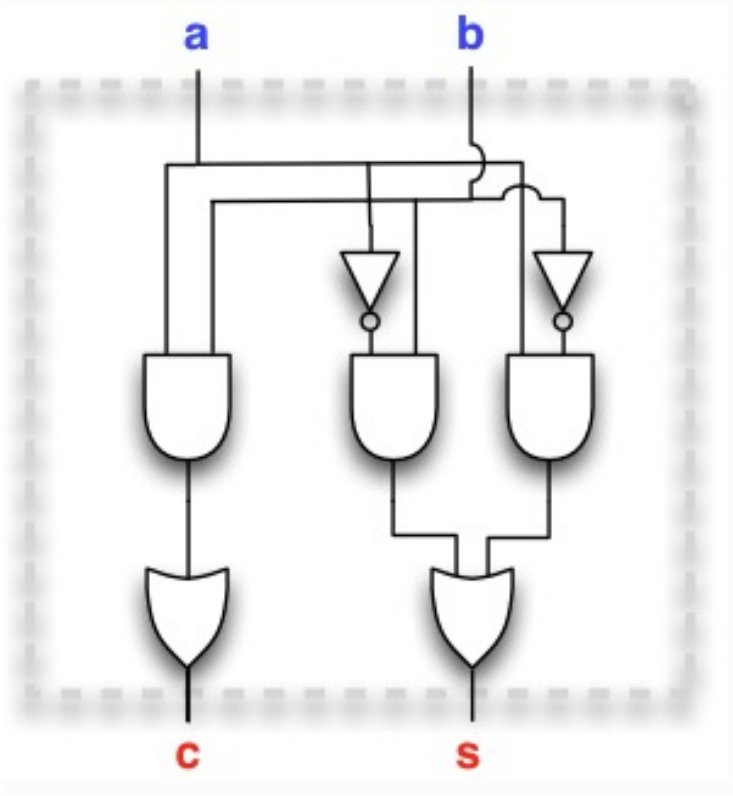


Olettaen että syöteet (a, b) on `tinylogGate`-olioita, Scala luo automaattisesti piirit näistä ilmiasuista!

No, ei tullut ihan sama…

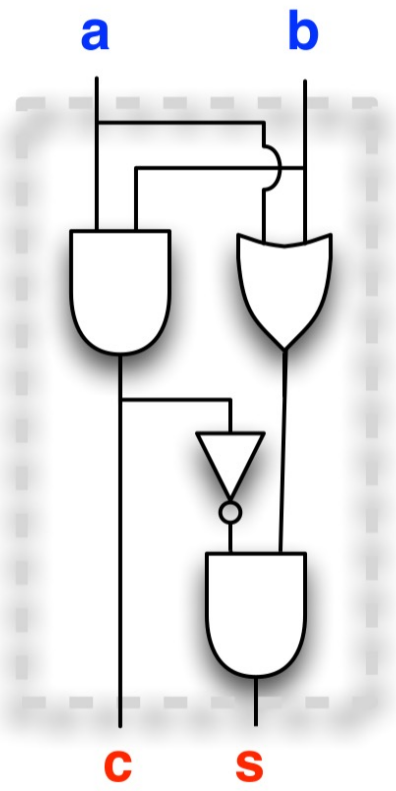
https://presemo.aalto.fi/o2fi2026

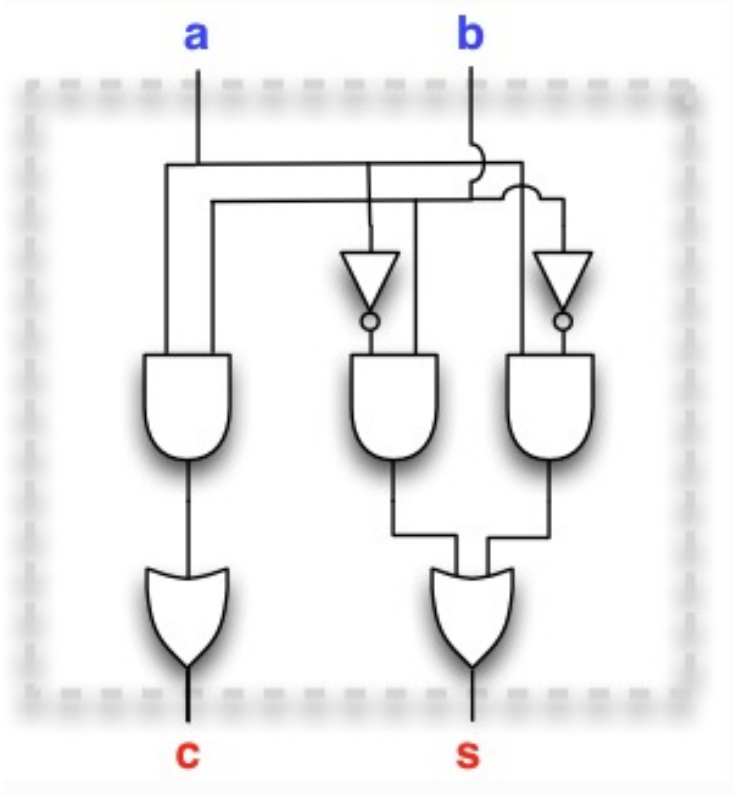




- **Kumpi on parempi ja miksi?**

No, ei tullut ihan sama…



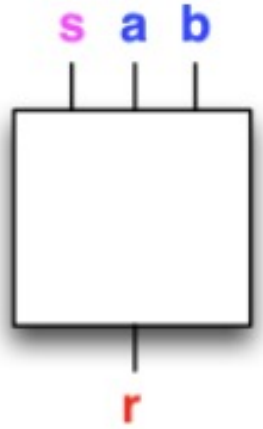


- **Erilaiset piirit voi tehdä saman asian**
  - Yhtä nopeasti (yhtä syvä)
  - Isompi piiri vie enemmän energiaa ja materiaalia

Toinen esimerkki: valitsin/kytkin (selector, multiplexer)

- **Oletetaan, että haluamme valita kahden bitin `a` ja `b` välillä käyttäen valitsin-bittiä `s`**
  - Jos `s = 0`, haluamme tulokseksi `r` arvon, joka on `a`:lla
  - Jos `s = 1`, haluamme `b`:n arvon

s	a	b	r	
0	0	0	0	
0	0	1	0	
0	1	0	1	(!s && a && !b)
0	1	1	1	(!s && a && b)
1	0	0	0	
1	0	1	1	(s && !a && b)
1	1	0	0	
1	1	1	1	(s && a && b)




```
val r = (!s &&  a && !b)||
        (!s &&  a &&  b)||
        ( s && !a &&  b)||
        ( s &&  a &&  b)
```

Olemmeko valmiita?

- **Syntetisoi piiri kahden n-bittisen luvun yhteenlaskuun!**

- **Yleinen toimintatapa:**
  1. Rakenna totuustaulu
  2. Rakenna kaava
- **Voisimme, teoriassa, listata summan kaikki mahdolliset yhdistelmät syötteille ja ulostuluille**
  - Onko tässä ongelmia?

https://presemo.aalto.fi/o2fi2026

Ongelmia…

- **Jos yritämme rakentaa koneen, joka laskee yhteen kaksi 64-bittistä kokonaislukua?**
  - 2 x 64 = 128 syötettä
  - Totuustaulussa on tällöin 2^{128} = 340282366920938463463374607431768211456 riviä
  - Niistä pitäisi etsiä tulokseltaan ykköset, määritellä AND-lausekkeet, ja yhdistää ne OR-lausekkeilla
  - Uuuuh, tarvitsemme fiksumman tavan…

Monimutkaisempien piirien järkevämpi rakennustapa

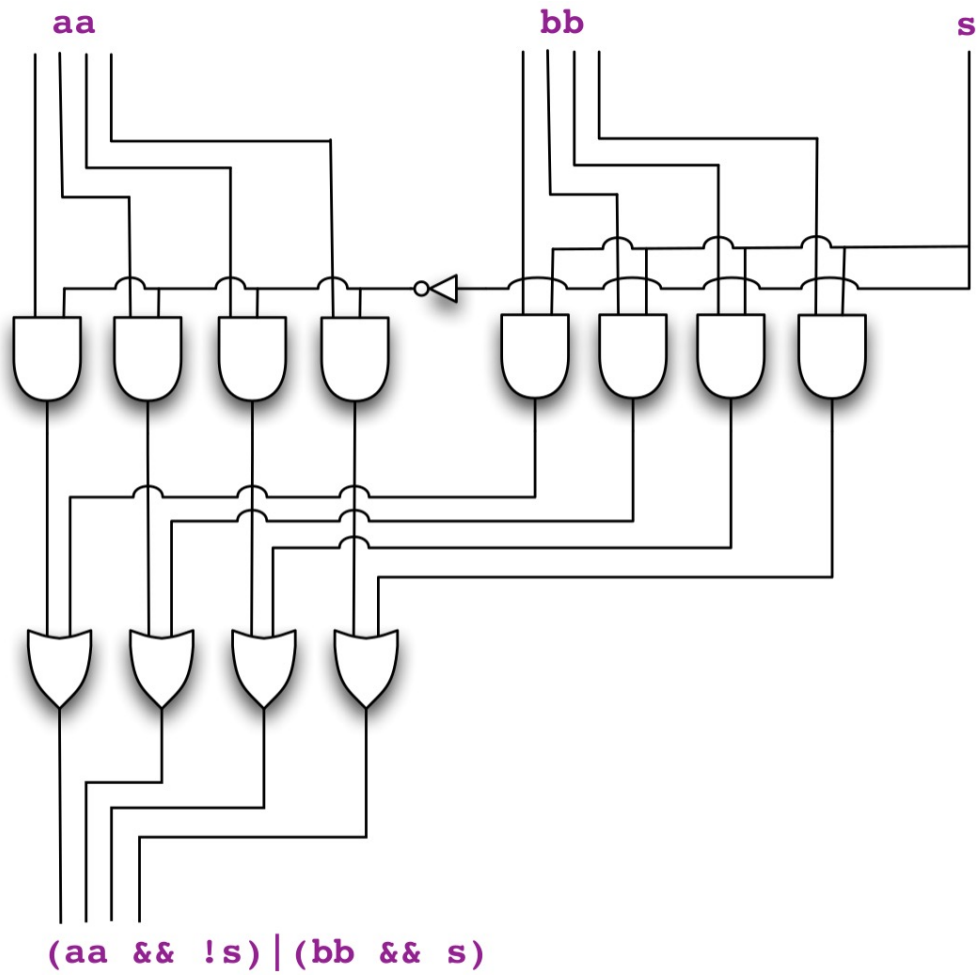
- **Portit käsittelevät vain yhtä bittiä**
- **Datan esittämiseen tarvitsemme useita bittejä**

- **Porttisekvenssejä kutsutaan väyläksi (bus)**
  - Väylä kuljettaa useiden bittien sanaa yhden bitin johdinten sijaan
- **Käyttäen modulaarista lähestymistapaa voi saada oivalluksia käsillä olevan tehtävän parempaan piirisuunnitteluun**

`Bus`-luokan implementaatio `tinylogissa`

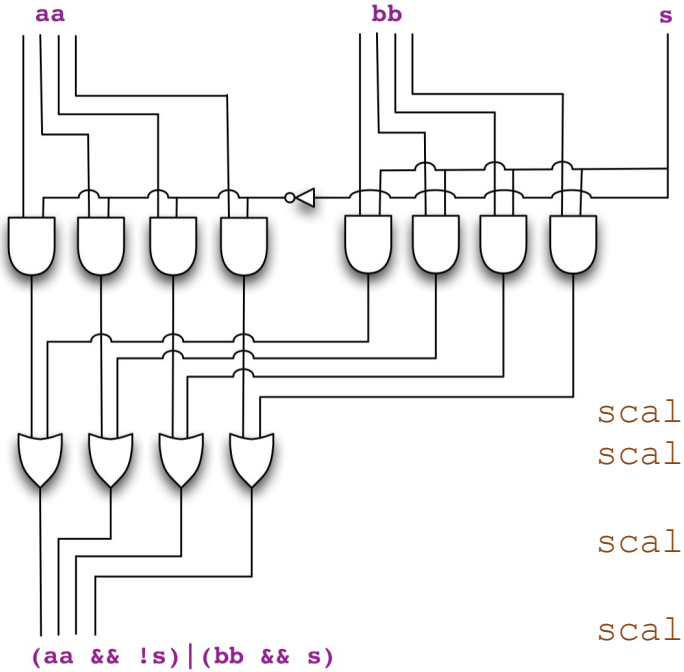
- **`Bus` eli väylä esittää useita bittejä (portteja), joten tehdään se laajentamalla Scalan `Seq`-piirreluokkaa**
  - `Bus` on siis mukautettu Scala-kokoelma `Gate`-objekteille
  - Koska `Bus` on `Seq`, meillä on pääsy kokoelmien tavallisiin metodeihin, kuten `map` ja `fold`
- **Sille määritellään myös väylätason operaattorit AND, OR ja NOT (`&`, `|`, `~`)**
  - Muista sana-operaatioiden viimeinen kierros
- **Kurssimateriaalissa on lisää `Bus`-luokan rakentamisesta**
  - Tämän kierroksen `tinylog`-paketista löytyy `bus.scala`

Rakennetaan valitsin väylälle



- Piiri, joka antaa väylän
  - `aa` arvon, jos `s` on `0` ja
  - `bb`:n, jos `s` on `1`

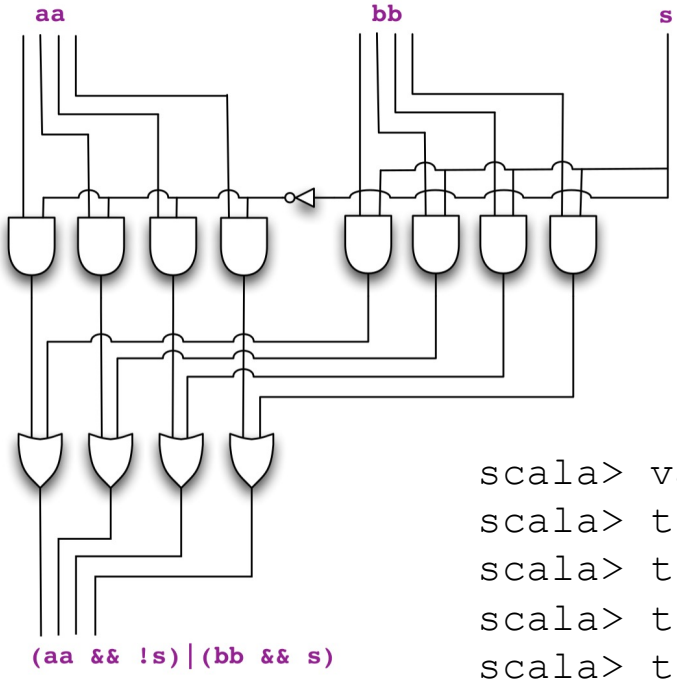
Rakennetaan valitsin väylälle



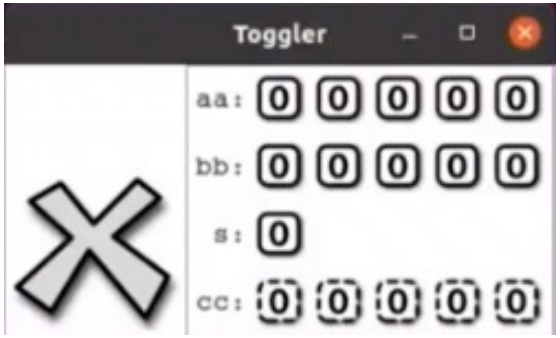
- Piiri, joka antaa väylän
  - `aa` arvon, jos `s` on `0` ja
  - `bb`:n, jos `s` on `1`
- Scalalla:

```
scala> import tinylog.*_
scala> val aa = Bus(Gate.False, Gate.True, Gate.True, Gate.True)

scala> val bb = Bus(Gate.True, Gate.False, Gate.False, Gate.False)

scala> val s = Gate.input() // Use an input gate so we can select

scala> val cc = (aa && !s) | (bb && s) // Construct the circuit

scala> cc.values // By default s is false, so we will select aa
val res0: Seq[Boolean] = List(false, true, true, true)

scala> s.set(true) // But, by setting s to true...

scala> cc.values // The circuit output has changed to bb
val res2: Seq[Boolean] = List(true, false, false, false)
```

Rakennetaan valitsin väylälle – käyttöliittymä

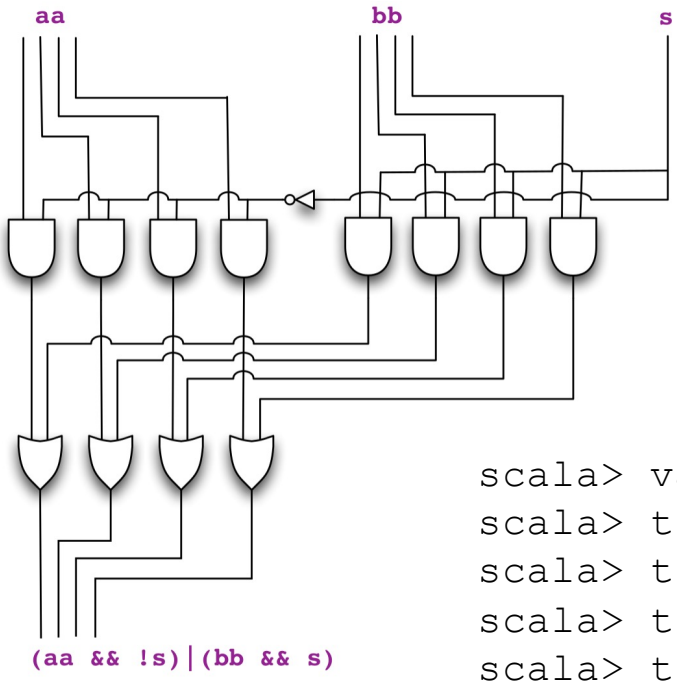


- **`Tinylog`-paketista löytyy myös `Toggler`, joka näyttää käyttöliittymän**

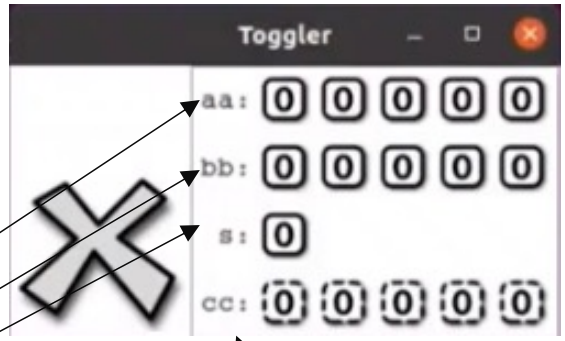



```
scala> val t = new Toggler() // Create toggler UI
scala> t.watch("aa", aa)     // Display bus aa
scala> t.watch("bb", bb)     // Display bus bb
scala> t.watch("s", s)       // Display selector bit
scala> t.watch("cc", cc)     // Display output bus cc
scala> t.go()                // And start
```

Rakennetaan valitsin väylälle – käyttöliittymä



- **`Tinylog`-paketista löytyy myös `Toggler`, joka näyttää käyttöliittymän**



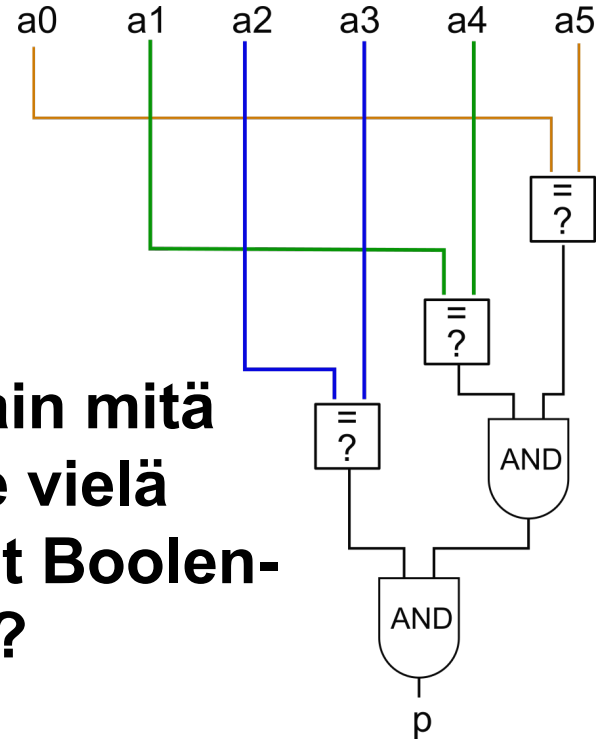

```
scala> val t = new Toggler() // Create toggler UI
scala> t.watch("aa", aa)     // Display bus aa
scala> t.watch("bb", bb)     // Display bus bb
scala> t.watch("s", s)       // Display selector bit
scala> t.watch("cc", cc)     // Display output bus cc
scala> t.go()                // And start
```

- Voit vaihtaa syötebittien arvoja ja katsoa, mitä tapahtuu ulostulolle

Demovideo:
https://aalto.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=32c4886a-3618-4c42-a251-b27400776790

Esimerkki: palindromi

- **Funktio joka rakentaa piirin, joka huomaa, jos `Bus` sisältää bitti-palindromin**
- **Piiri kannattaa ensin hahmotella kuvaamalla, mitä haluamme sen tekevän**
  1. Tarkista onko ensimmäinen portti sama kuin viimeinen
  2. Tarkista onko toinen portti sama kuin toiseksi viimeinen
  3. …
  n. Jos kaikki nämä ovat totta, meillä on palindromi

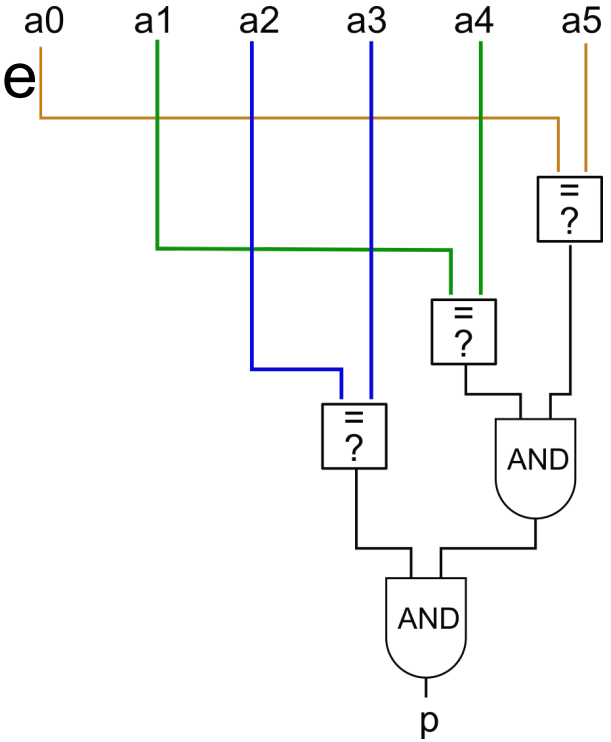


- **Onko jotain mitä emme ole vielä ilmaisseet Boolen-logiikalla?**
  - Kyllä: yhtäsuuruuden tarkistus: `=?`
  - Tiedämme miten!
    `(a && b) || (!a && !b)`

Esimerkki: palindromi

- Huom. Kun rakennamme piirin, koodin ei tarvitse missään vaiheessa tarkistaa portin arvoa!

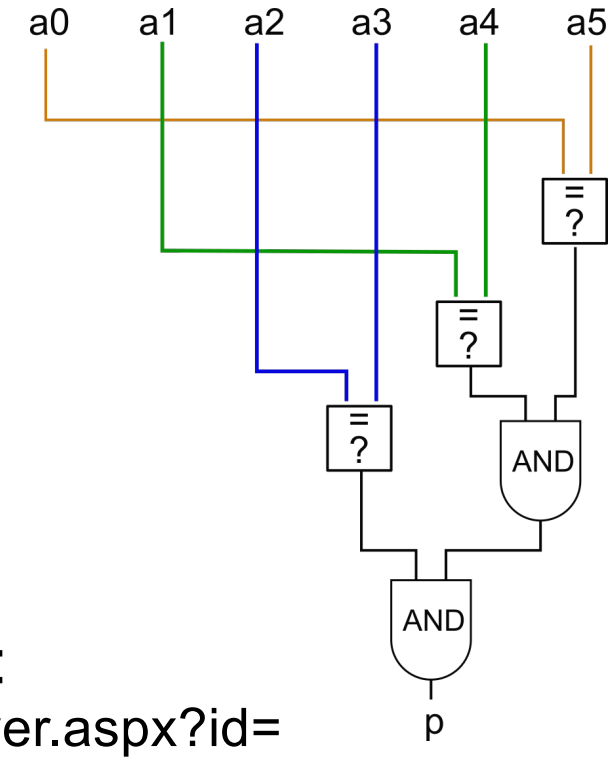
```
// Helper function - output gate is true when input
// gates are equal
def gatesEqCircuit(a : Gate, b : Gate) : Gate =
  (a && b) || (!a && !b)

// Output gate is true when input bus is a palindrome
def isPalindromeCircuit(aa: Bus) : Gate =
  val len = aa.length
  // Create an array of size len/2 to hold the eq gates
  val eqc = new Array[Gate](len/2)
  for i <- 0 until len/2 do
    eqc(i) = gatesEqCircuit(aa(i), aa(len-1-i))
  end for
  // Now we can just reduce it with AND to form the
  // final gate
  eqc.reduce(_ && _)
end isPalindromeCircuit
```



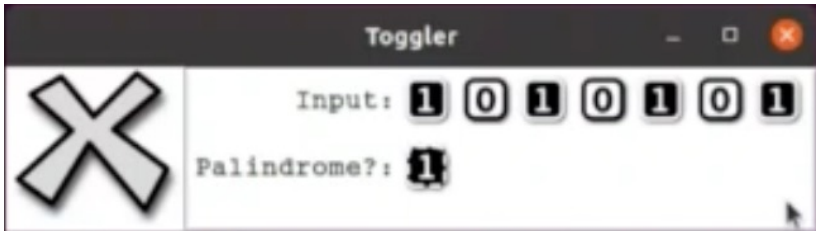
Esimerkki: palindromi

- Togglerissa käynnistäminen:

```
val aa = Bus.inputs(7)
val p = isPalindromeCircuit(aa)
val t = Toggler()
t.watch("Input",aa)
t.watch("Palindrome?",p)
t.go()
```



- Tämäkin löytyy Panoptosta (linkki MyCossa):
  https://aalto.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=f5bb39dd-cc39-4cc8-9395-b274007767d5



Mitä tekee se piiri, jonka `f` palauttaa?

https://presemo.aalto.fi/o2fi2026

```
import tinylog.*

def e(a:Gate, b:Gate) = (a && b) || (!a && !b)

def f(aa:Bus, bb:Bus) =
  require(aa.length == bb.length)
  val x = aa.zip(bb).map((a,b) => e(a,b))
  x.reduce(_ && _)
end f
```

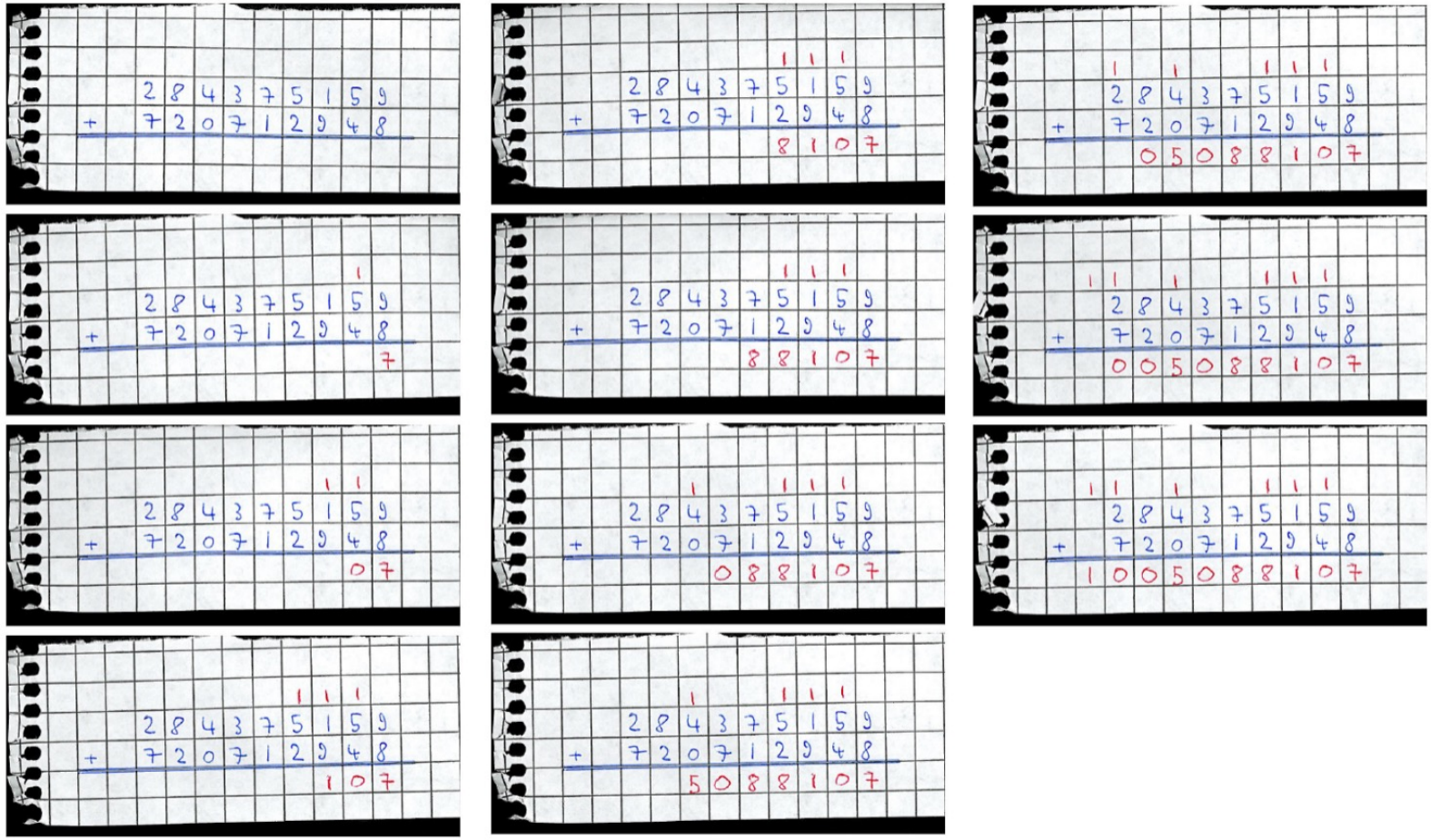
Ihan kiva, mutta…

Mites se 64-bittinen summain kahdelle luvulle?

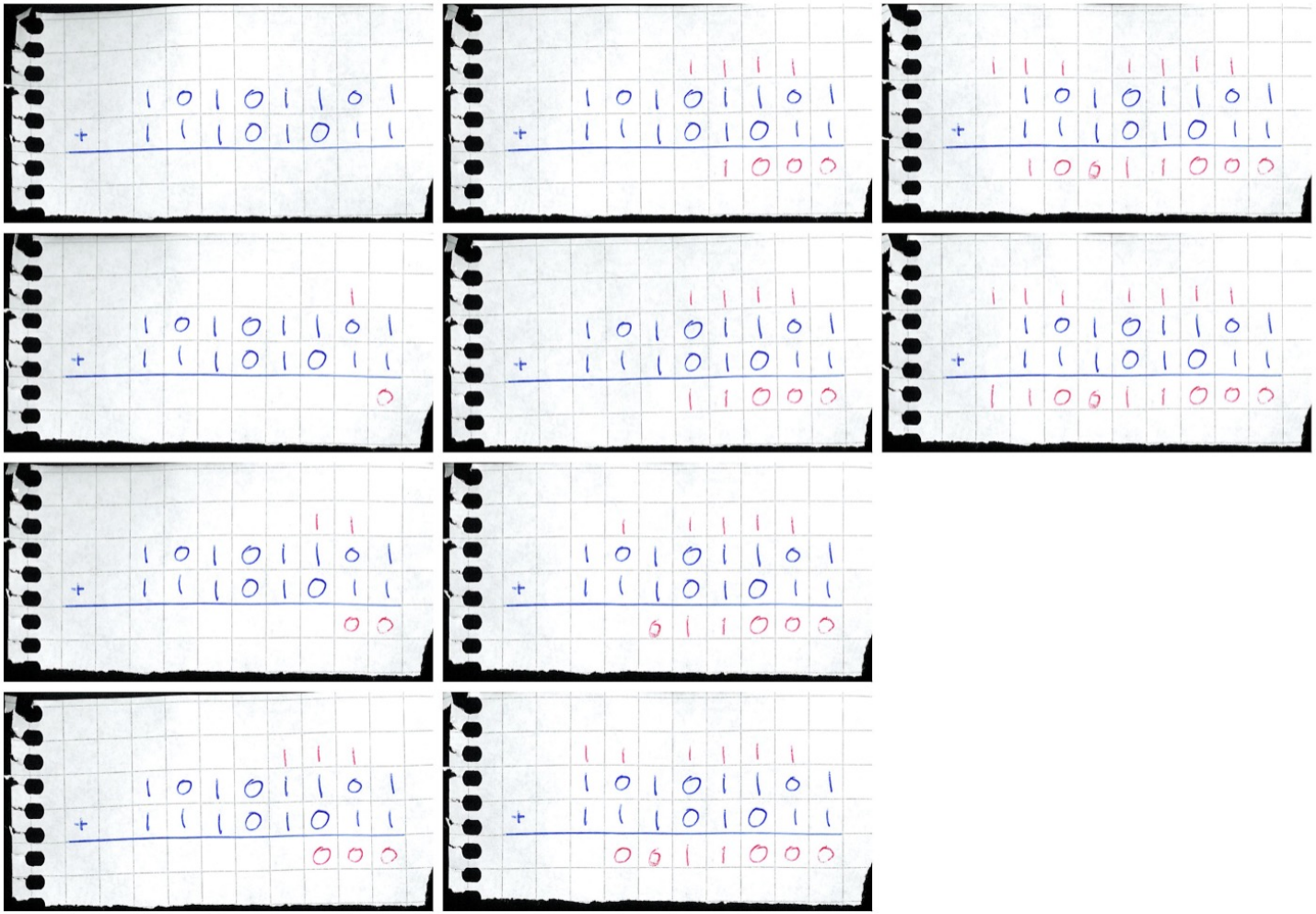
- **Unohdetaan tässä vaiheessa negatiiviset luvut…**

- **Mitä me teemme, kun laskemme yhteen kaksi kokonaislukua?**

Takaisin kouluun: 10-kantaiset luvut



Takaisin kouluun: 2-kantaiset luvut



Binäärilukujen yhteenlasku on helppoa!

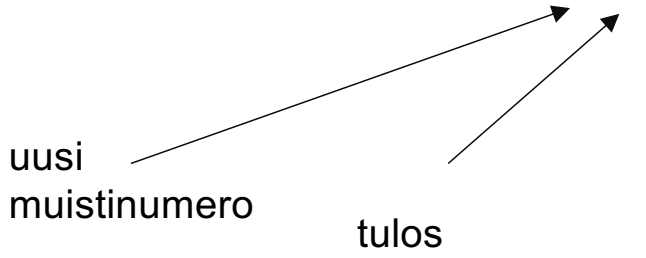
- **Rakenna piiri, joka seuraa menetteylä, yksi merkitsevä bitti kerrallaan**
- **Jokainen merkitsevä numero on määritetty laskemalla yhteen kolme numeroa:**
  - Syöte `a`
  - Syöte `b`
  - Muistinumero `c`
  - Jokainen niistä on joko `0` tai `1`

```
0 + 0 + 0 = 00
0 + 0 + 1 = 01
0 + 1 + 0 = 01
0 + 1 + 1 = 10
1 + 0 + 0 = 01
1 + 0 + 1 = 10
1 + 1 + 0 = 10
1 + 1 + 1 = 11
```

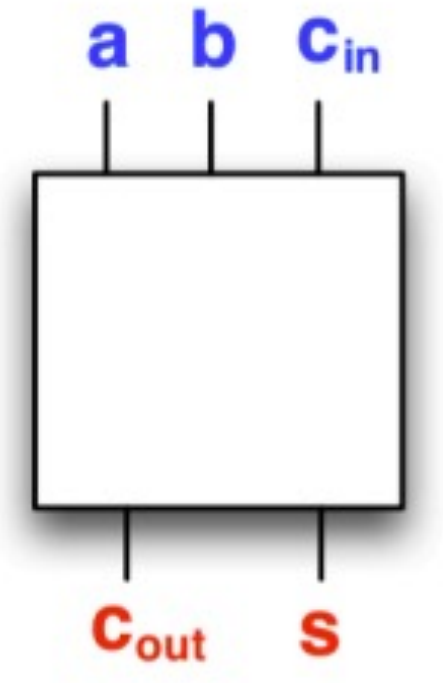
Binäärilukujen yhteenlasku on helppoa!

- **Rakenna piiri, joka seuraa menetteylä, yksi merkitsevä bitti kerrallaan**
- **Jokainen merkitsevä numero on määritetty laskemalla yhteen kolme numeroa:**
  - Syöte `a`
  - Syöte `b`
  - Muistinumero `c`
  - Jokainen niistä on joko `0` tai `1`

```
a   b   c
0 + 0 + 0 = 00
0 + 0 + 1 = 01
0 + 1 + 0 = 01
0 + 1 + 1 = 10
1 + 0 + 0 = 01
1 + 0 + 1 = 10
1 + 1 + 0 = 10
1 + 1 + 1 = 11
```



Kokosummain (full adder) -piiri



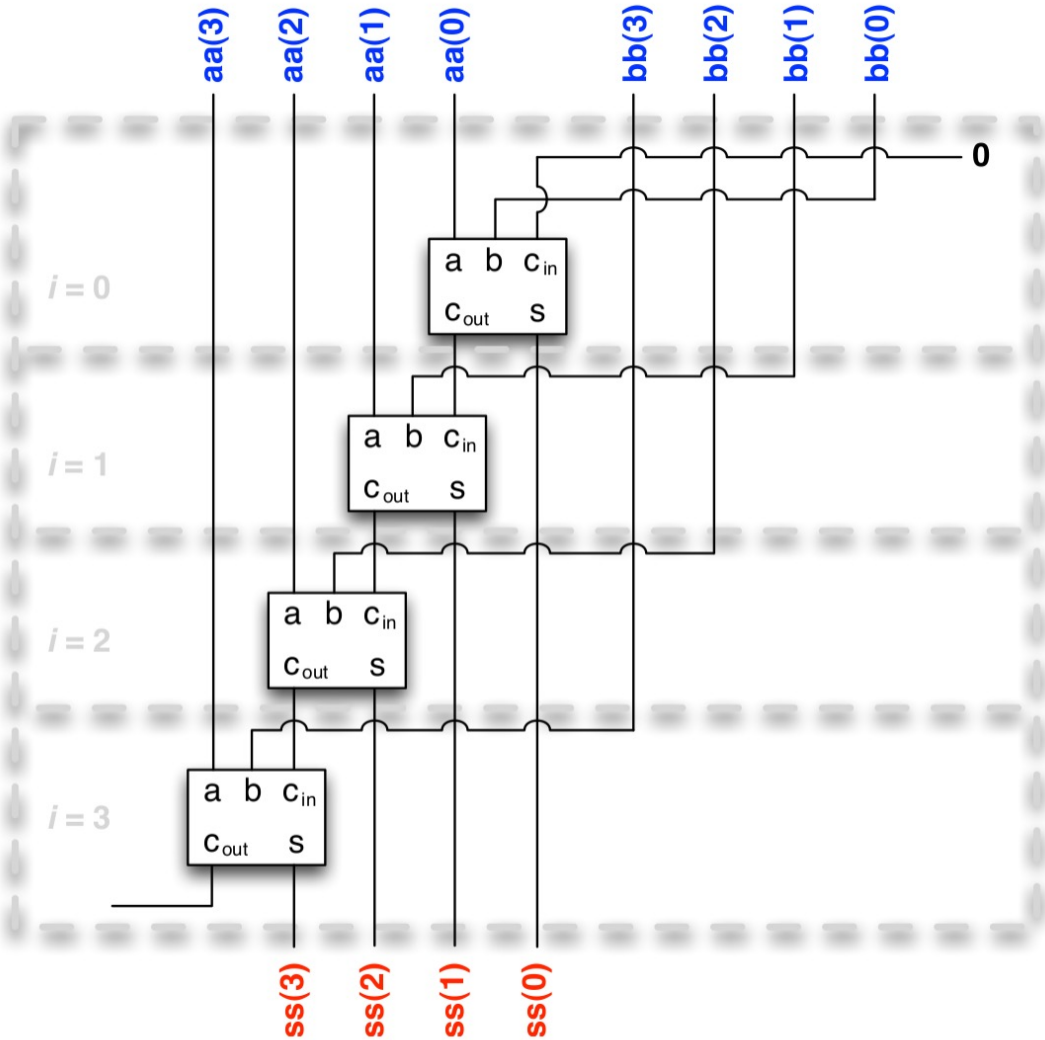
a	b	c_{in}	c_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

a	b	c_{in}	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

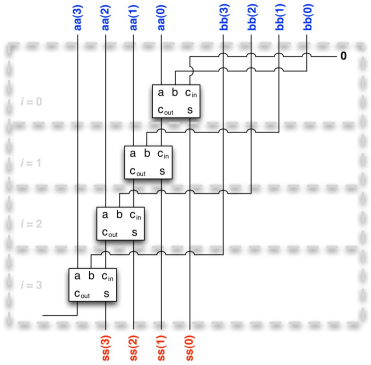
- `c` on muistinumero (carry)

Ketjutettu summain (4-bittiselle koneelle)

- Yhdistää kokosummaimia yksi merkitsevä bitti kerrallaan
- Suomenkieliseltä nimeltään **ketjutettu summain**
- Englanniksi **ripple-carry adder** koska bitit leviävät kuin väreily veden pinnalla



4-bittinen ketjutettu summain – toteutus




```
def buildFullAdder(a: Gate, b: Gate, c_in: Gate): (Gate, Gate) =
   val c_out = (!a && b && c_in) || (a && !b && c_in) || // ...
     (a && b && !c_in) || (a && b && c_in)
   val s = (!a && !b && c_in) || (!a && b && !c_in) || // ...
     (a && !b && !c_in) || (a && b && c_in)
   (c_out, s)
 end buildFullAdder

 def buildRCAdder(bus1: Bus, bus2: Bus) =
   require(bus1.length == bus2.length,
     "Can only add buses of same length")
   var carry_in: Gate = Gate.False // no initial carry
   val ss = new Array[Gate](bus1.length)
   for i <- 0 until bus1.length do
     val (carry_out, sum) = buildFullAdder(bus1(i), bus2(i),
       carry_in)
     carry_in = carry_out // carry from bit i propagates
     ss(i) = sum
   end for
   new Bus(ss.toIndexedSeq)
 end buildRCAdder
```

Kun tulkitaan `Bus` datasanana, huomaa tämä!

- **Sekvenssien indeksien yleensä ajatellaan kasvavan vasemmalta oikealle, mutta (binääri)numerot indeksoidaan (vähiten merkitsevästä luvusta alkaen) oikealta vasemmalle**
- **Esim, tämä kolmen portin väylä:**

```
val aa = Bus(Gate.True, Gate.True, Gate.False)
```

esittää binäärimerkkijonoa `011`

- **Pitäkää tämä mielessänne, kun debugaatte harjoitustehtäviä!**
  - Togglerissa voi kääntää sekvenssin: `aa.reverse`

Viikon harjoitukset

1. **Circuit quiz: täydennä totuustaulu**
2. **Tinylog quiz: kuinka testata porttien arvoja**
3. **Gate exrpessions: `Gate`:ssa kaksi perusoperaatiota**
4. **Operations on Circuits**
   - Operaatioita käyttäen piirejä
   - Huom: count true bits (laske tosi-bitit) ei voi toteuttaa samoin kuin kierroksella 2
   - Video selittämään etumerkittömän vähemmän kuin –operaation:

   https://aalto.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=fe5faa24-a0be-41eb-b8d7-b300008f98b8
5. **Bus writer**
   - Piiri joka valitsee väylän ja kirjoittaa dataa siihen
6. **(Haaste: Shallow operations)**

Vihjeitä viikon harjoituksiin

- **Piirien piirtäminen paperille helpottaa**
- **Kaikissa harjoituksissa rakennetaan piirejä `Gate`- ja `Bus`-olioista käyttäen Boolen-operaatioita**
- **`Gate`:n (`Bus`:in) arvoa ei tiedetä, kun sitä rakennetaan!**
  - Jos haluat tarkistaa `Gate`- tai `Bus`-olion arvon rakentaessasi piiriä, koodisi on todennäköisesti virheellinen
  - Käytä sen sijaan Boolen-operaattoreita niille. Nämä arvioidaan joka kerta uudelleen, kun syöte muuttuu!
- **Piirit ja väylät sisältävät *Play.scala ohjelman**
  - Runko käyttäen `Trigger`iä, jota voit editoida ja jotta voit kokeilla piirien toimintaa

Fukseille – Johdatus opiskeluun to 12.3.2026!

- **Muistakaa Johdatus opiskeluun –kurssin luento to 12.3.2026 klo 14:15**
  - Info: TU7, Maarintie 8: valinnat uran aikana
  - Tepsy: maisteriohjelman valinta myöhemmin
  - Tfy: U4, Otakaari 1: Valinnat uran aikana
  - Tik: T1, Konemiehentie 2, Akateeminen urapolku
  - Tuta, ei luentoa vaan omatoiminen tehtävä

- **Yhteinen sivuaineluento to 19.3 klo 14:15, Jeti, Otakaari 5**