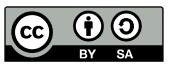
Flash storage technology

Arnout Vandecappelle
arnout@mind.be

Updated slides can be found at
http://mind.be/content/Presentation_Flash-technology-ELCE16.odp



Flash storage technology

- **Workarounds**
  to make flash work

Flash storage technology

- How flash **fails to work**
- **Workarounds**
  to make flash work

Flash storage technology

- How flash **works**
- How flash **fails to work**
- **Workarounds**
  to make flash work

Flash storage technology

- How flash **works**
- How flash **fails to work**
- **Workarounds**
  to make flash work

This is just my internet research

How flash works

This presentation is about:

- NAND flash
- MLC cell
- Small technology nodes (16nm)

NAND flash is cheap

Driver: $ per bit

- Developed for SD cards and SSD
  Chips we get for embedded are a by-product
- Compared to HDD, speed is not an issue
- Compared to HDD, power is not an issue
- Reliability is not measurable

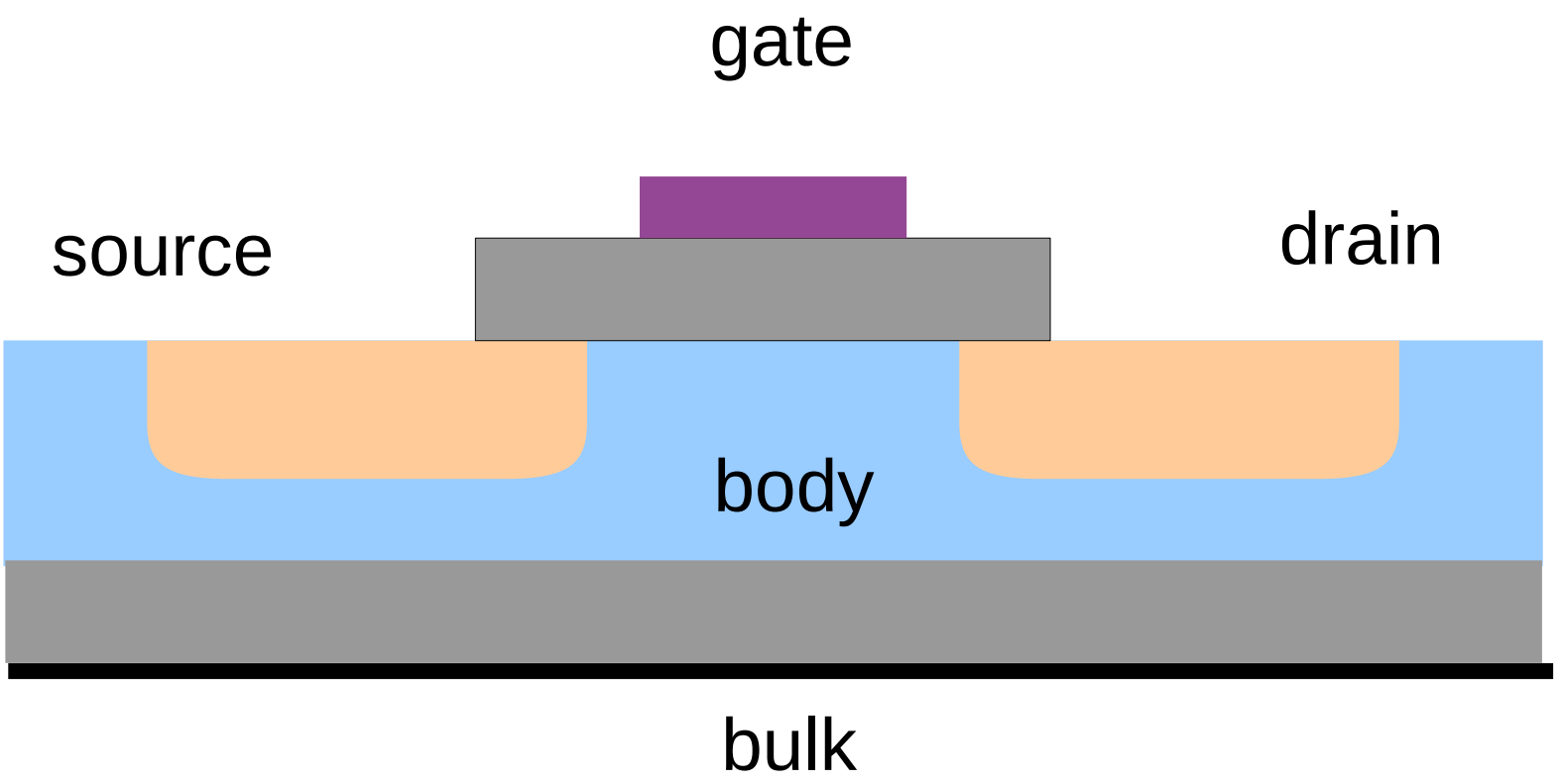
How flash works

- Basic cell structure
- Reading
- Programming
- Erasing

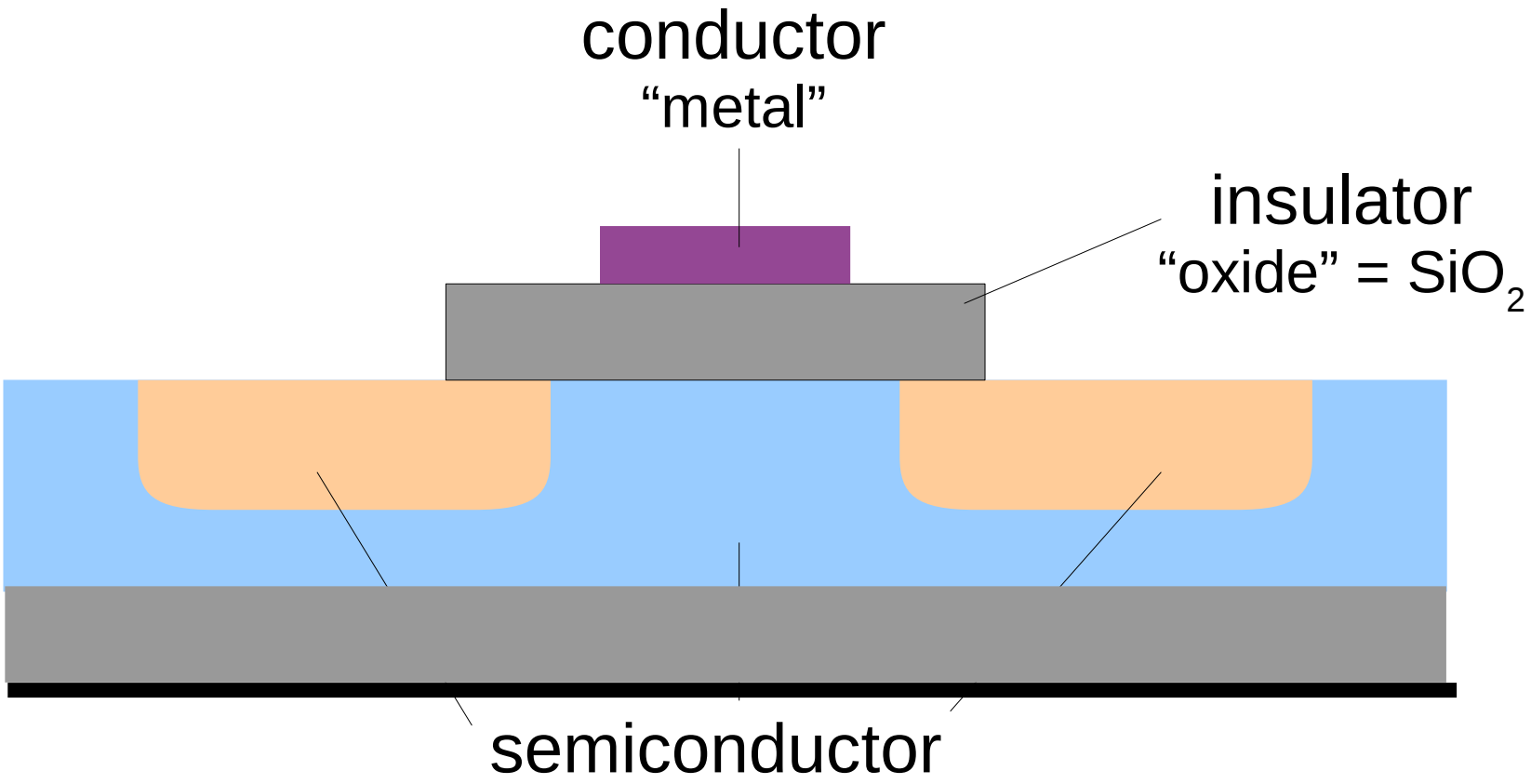
How flash works

The
Electrically
Erasable
Programmable
ROM
cell

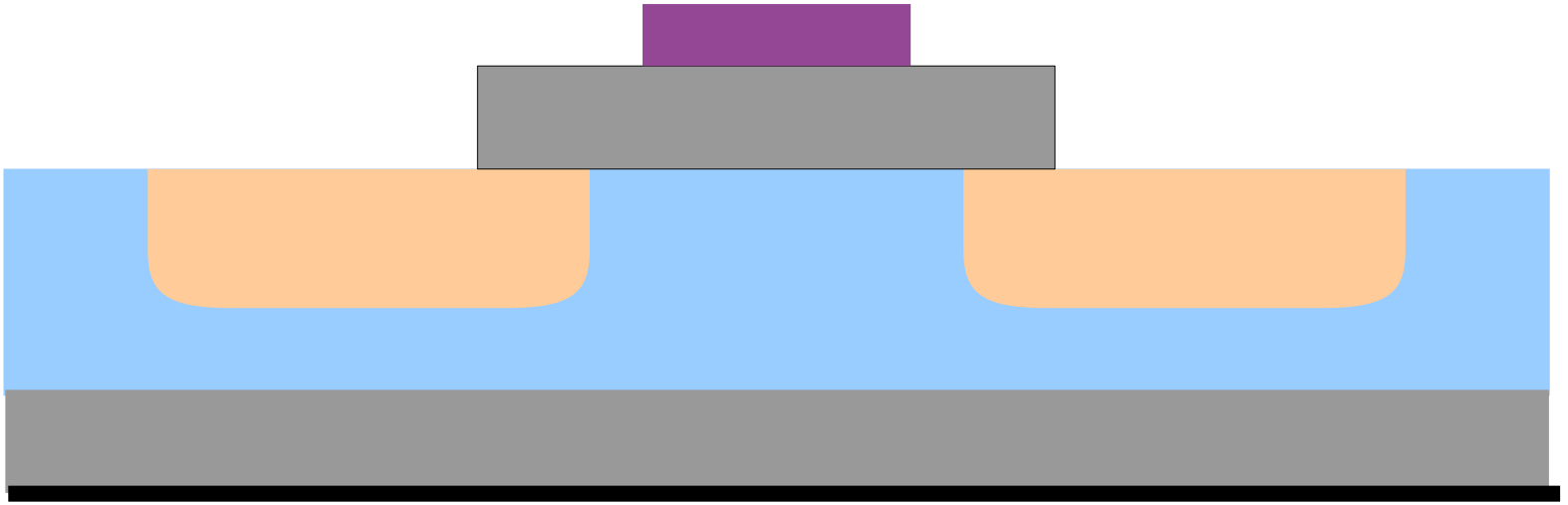
Back to basics
The transistor



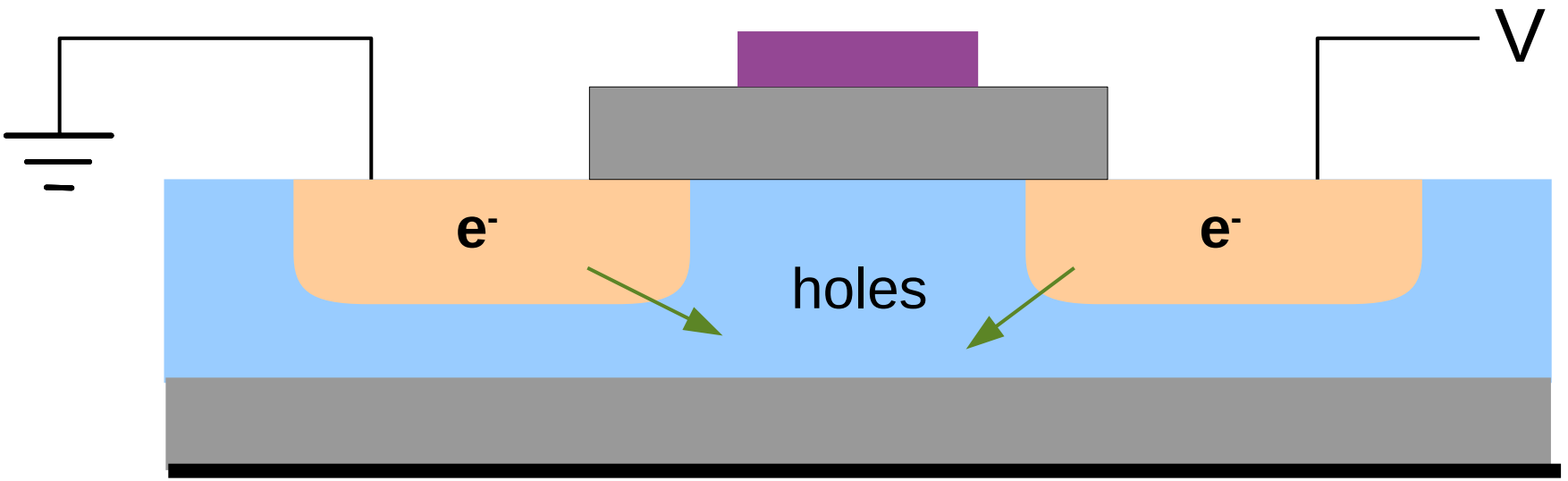
Back to basics
The transistor



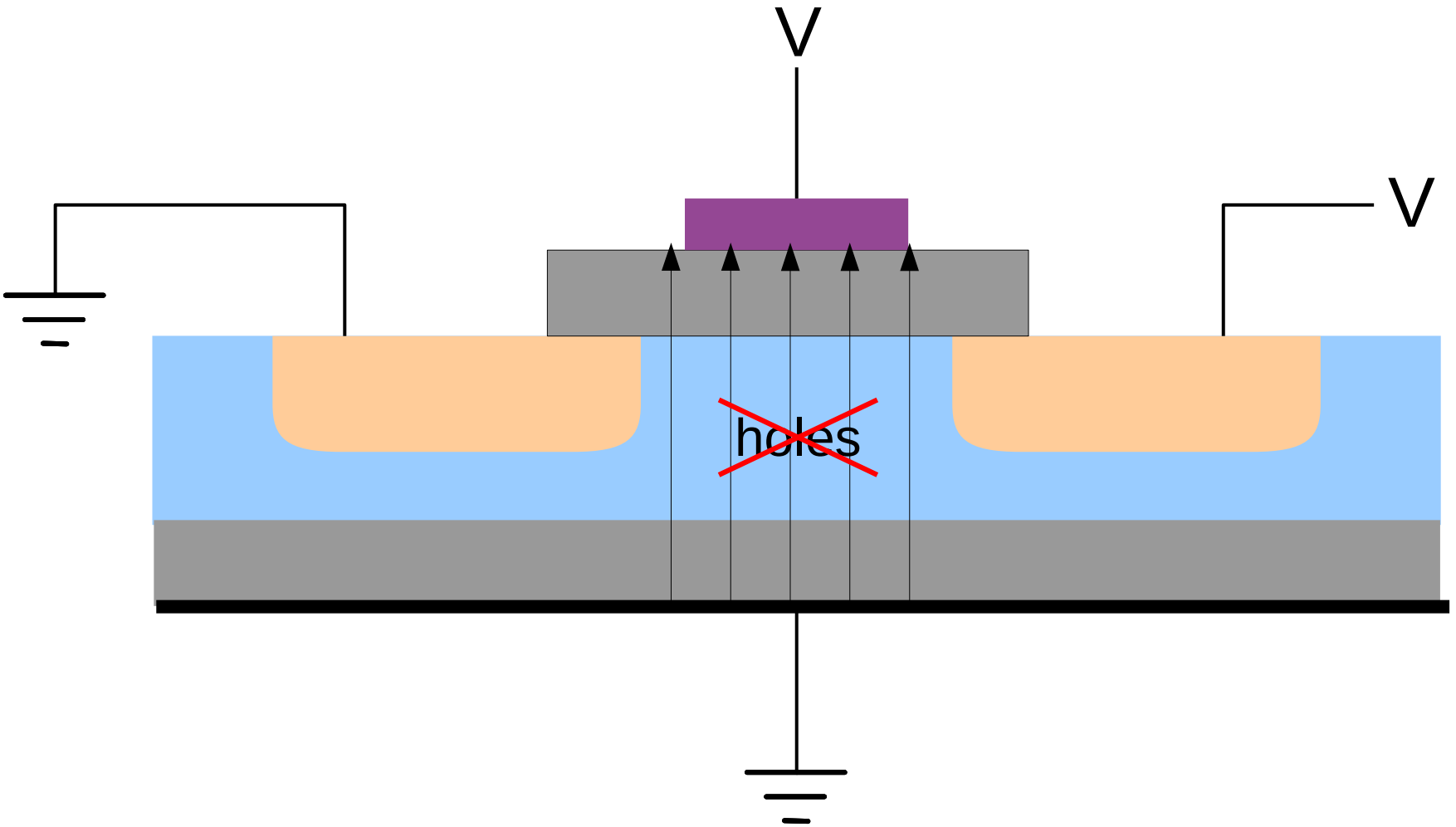
Back to basics
The transistor



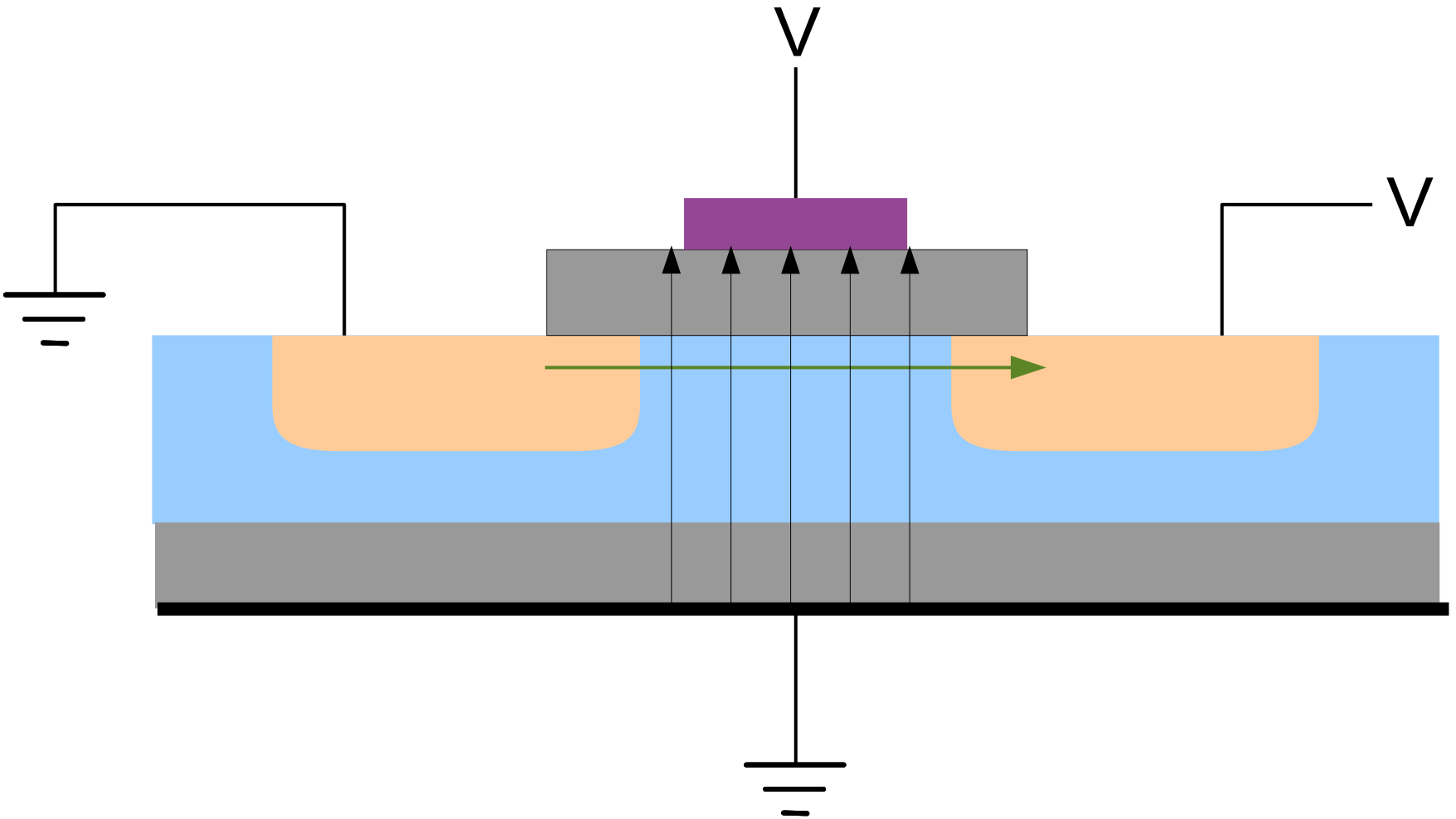
Back to basics
The transistor



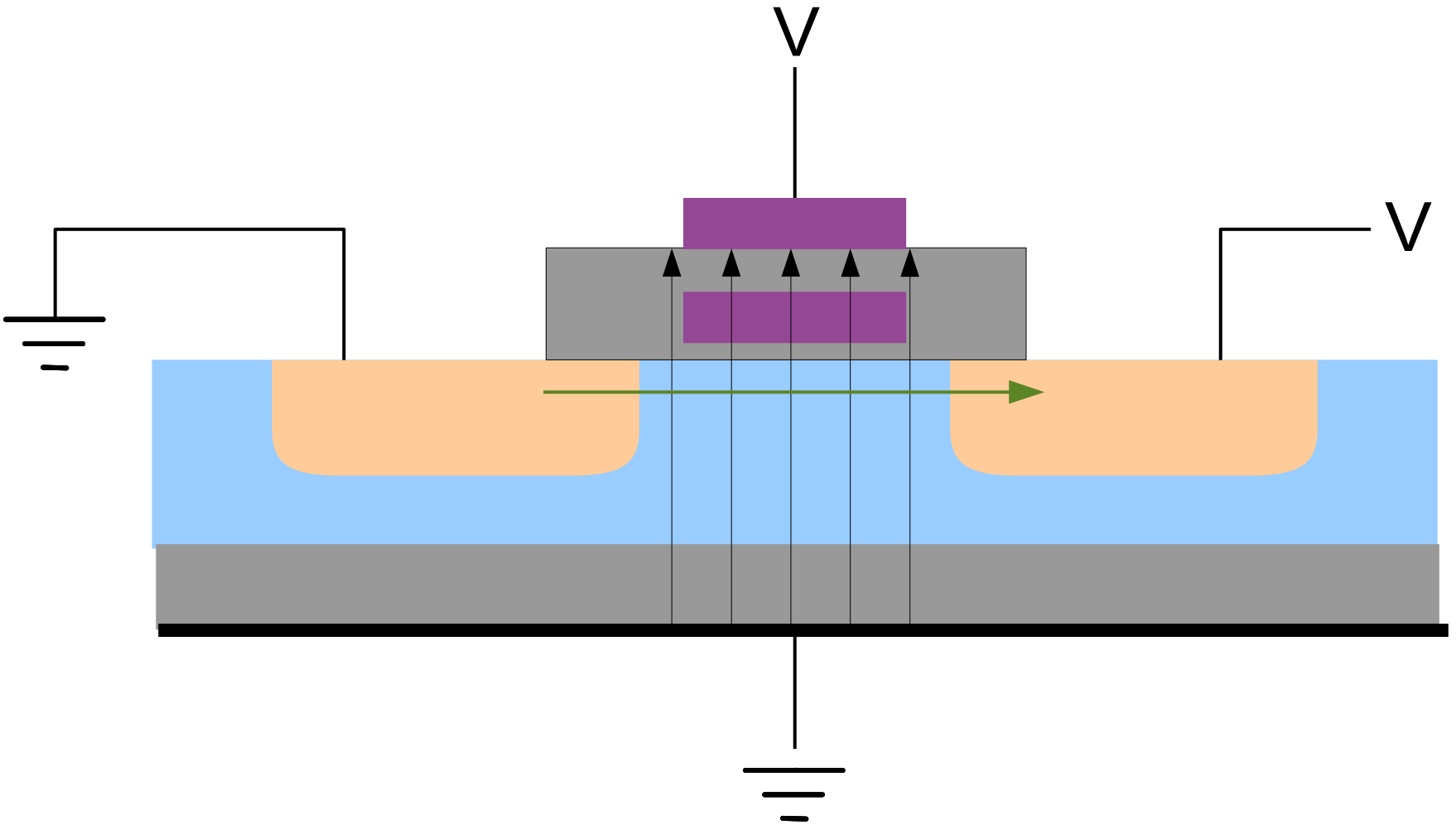
Back to basics
The transistor



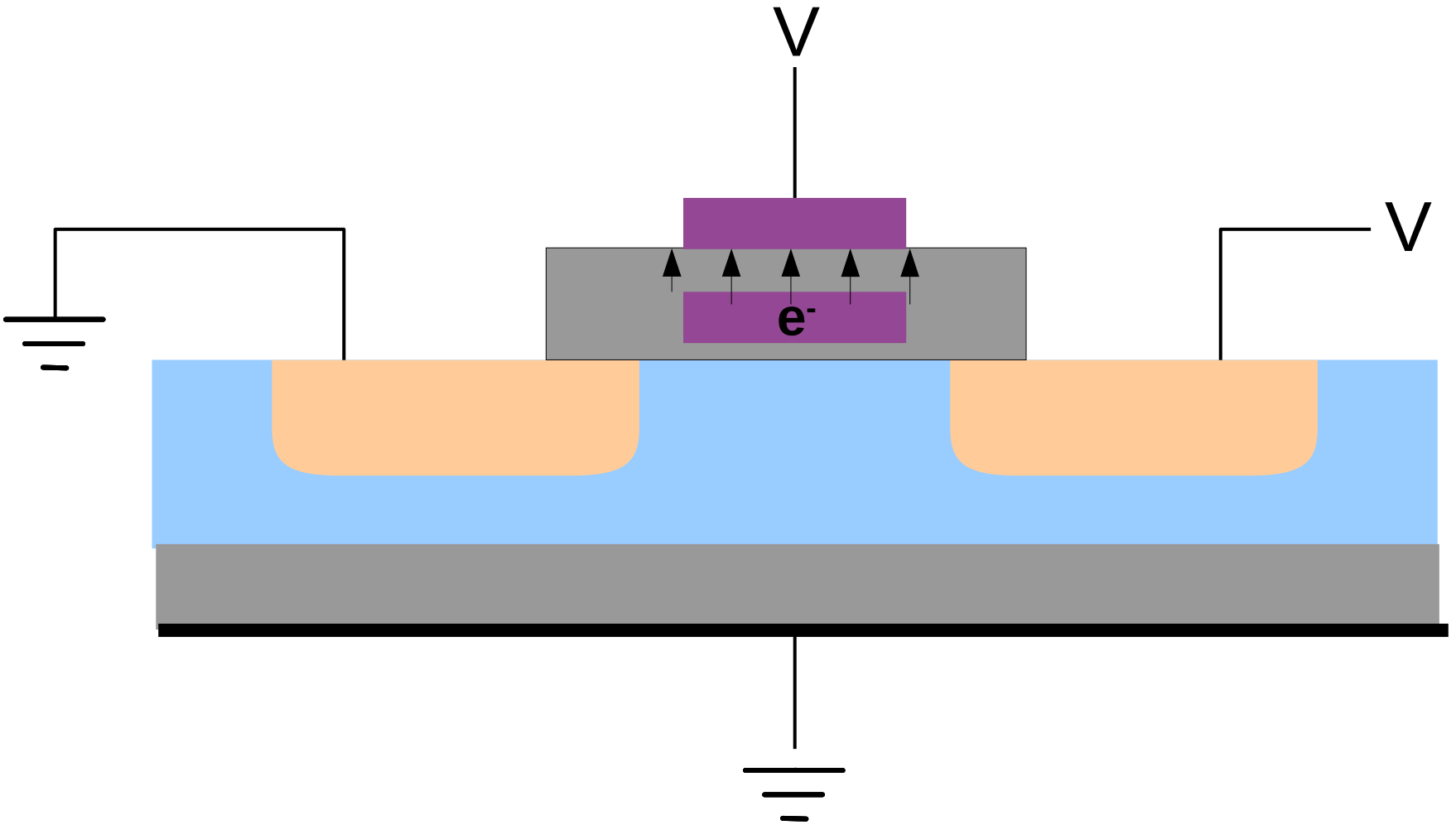
Back to basics
The transistor



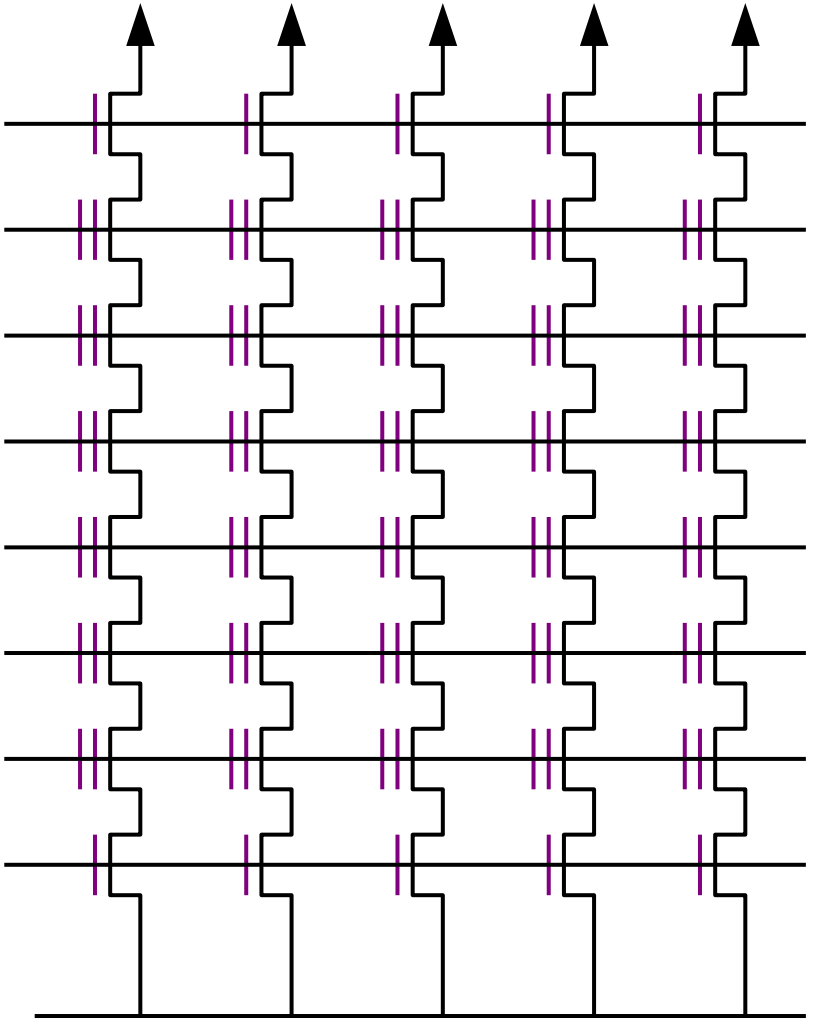
A transistor with memory



A transistor with memory



Combining cells in NAND

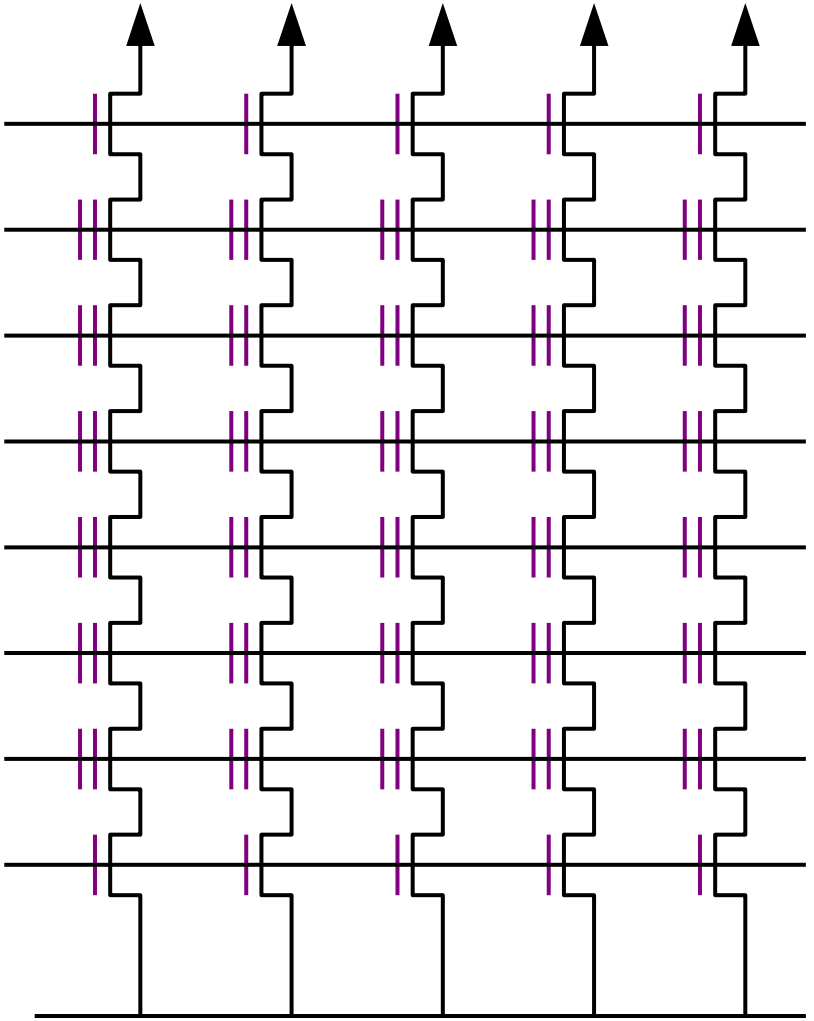


Extremely dense structure

Peripheral circuits optimised away

Easy to scale

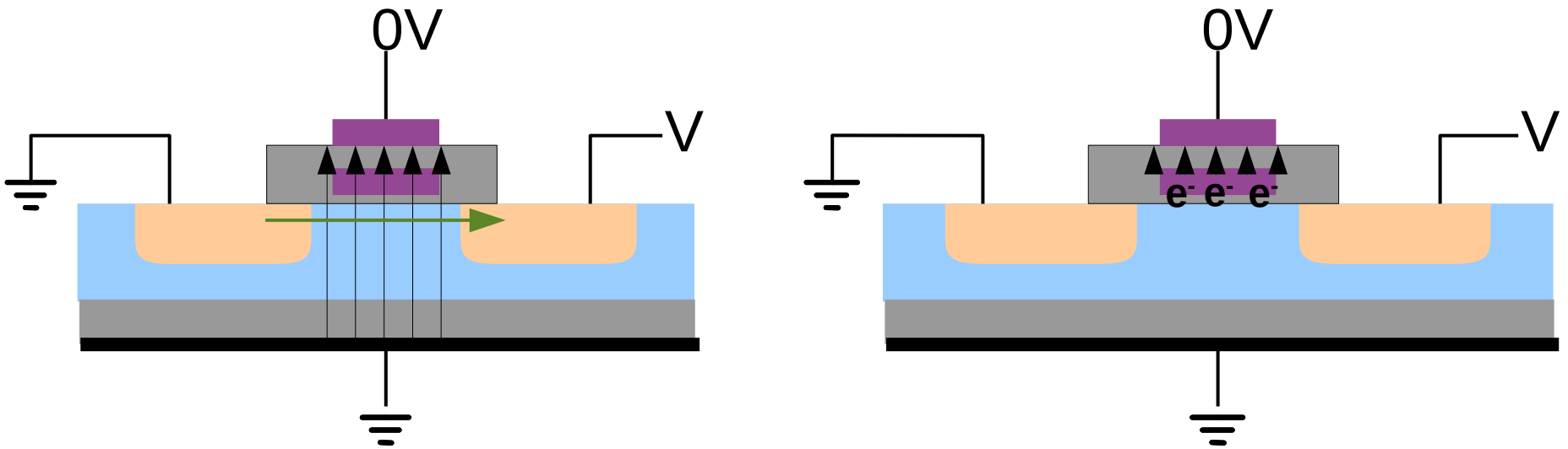
Combining cells in NAND



String of 32 cells

16896 strings
= 32 * 2112 bytes
= 32 pages

Increasing density: MultiLevel Cell (MLC)



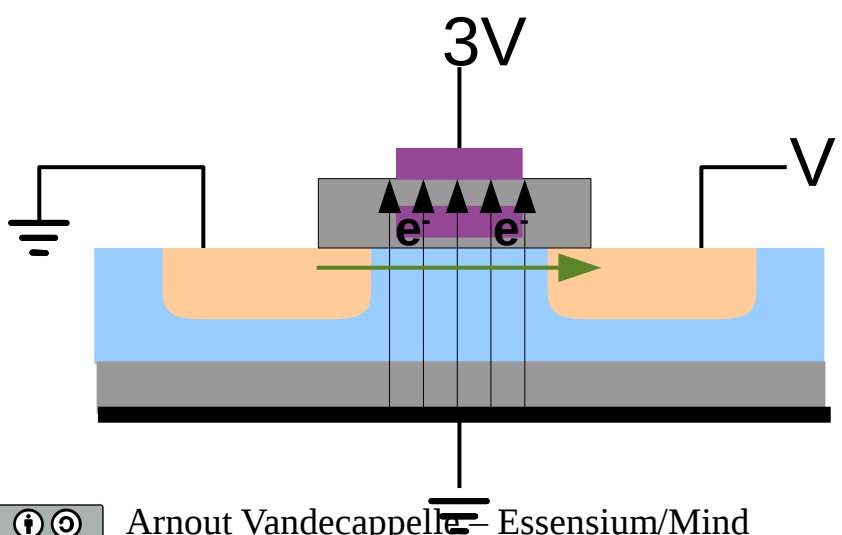
empty cell
passes @0V

full cell
blocks @0V

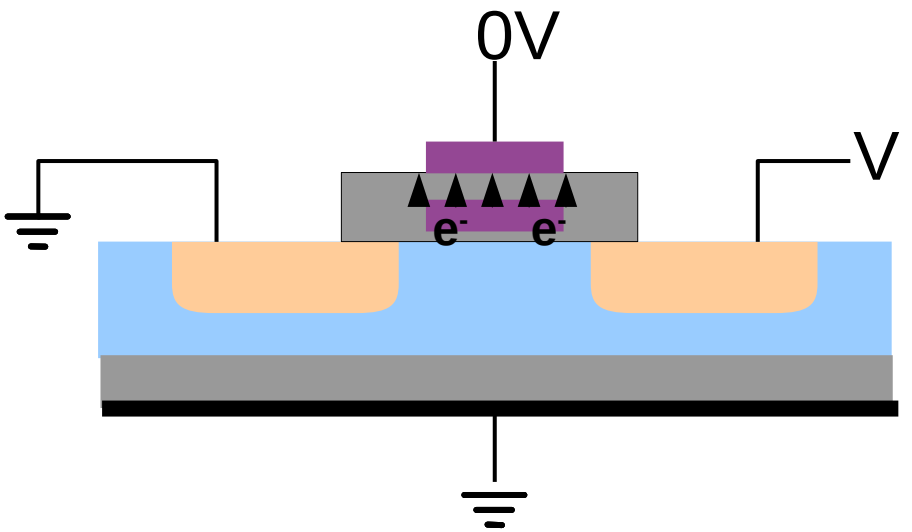
= Single Level Cell (SLC)
only 1 threshold voltage

Increasing density: MultiLevel Cell (MLC)

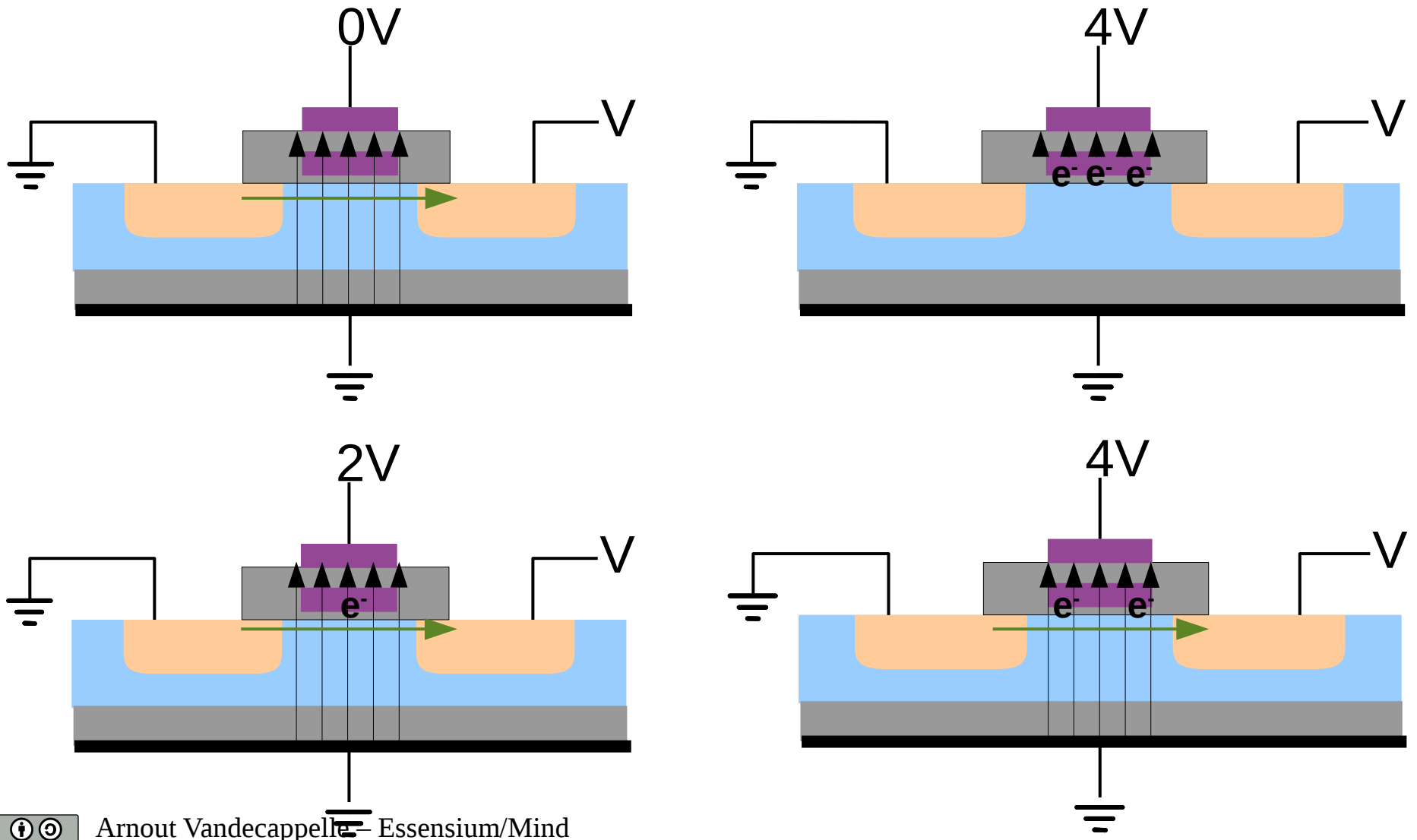
half cell
passes @3V



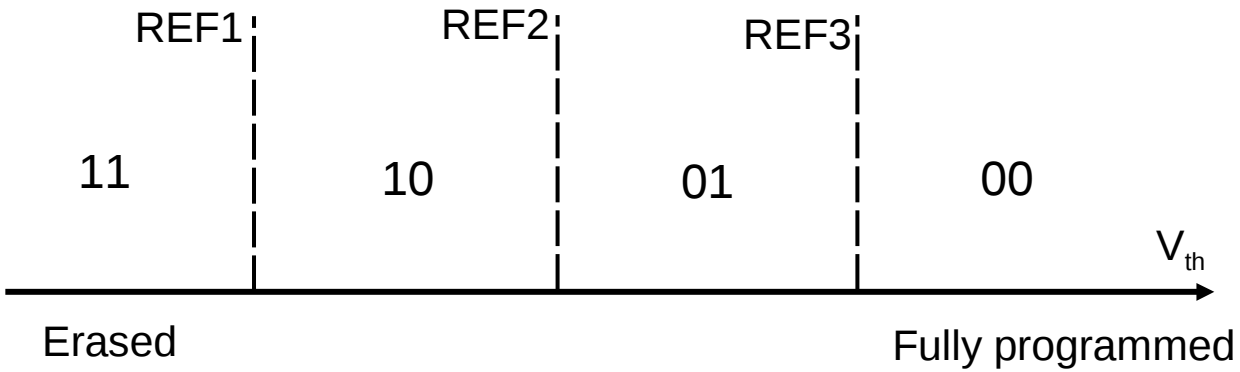
half cell
blocks @0V



Increasing density: MultiLevel Cell (MLC)

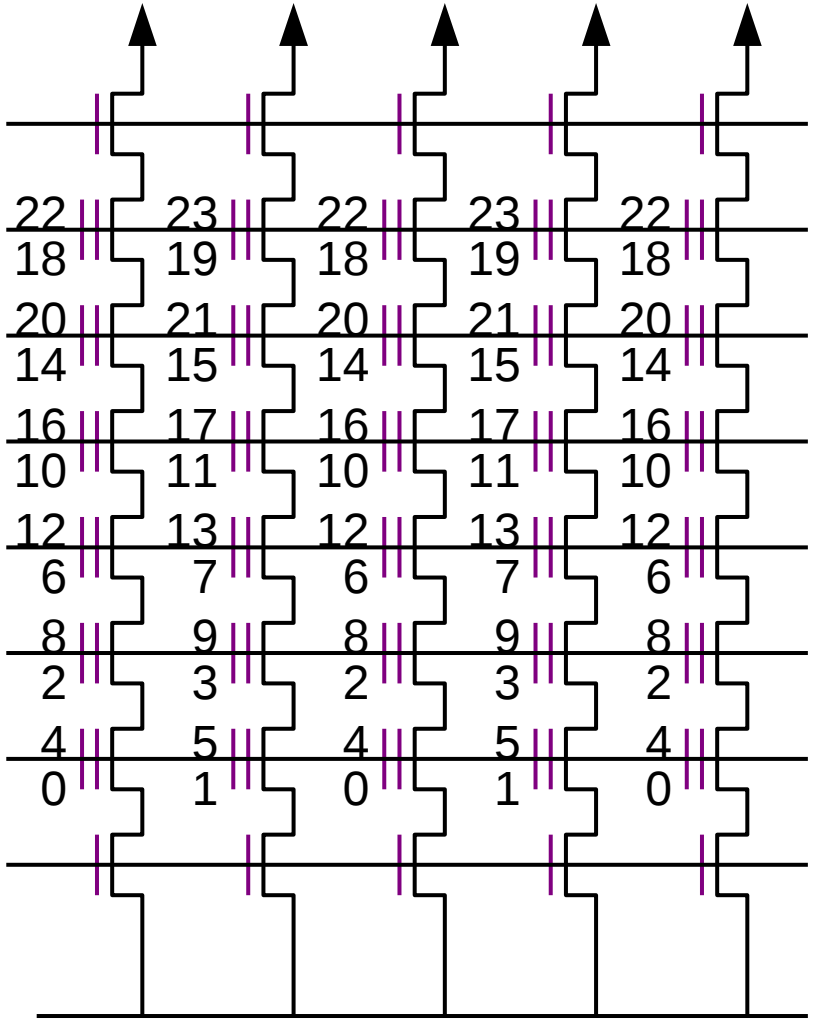


MultiLevel Cell has several threshold voltages



2 bits	3 levels
3 bits	7 levels
4 bits	15 levels
5 bits	31 levels
6 bits	63 levels

MLC pages are often interleaved



Odd/even pages
on separate bitlines

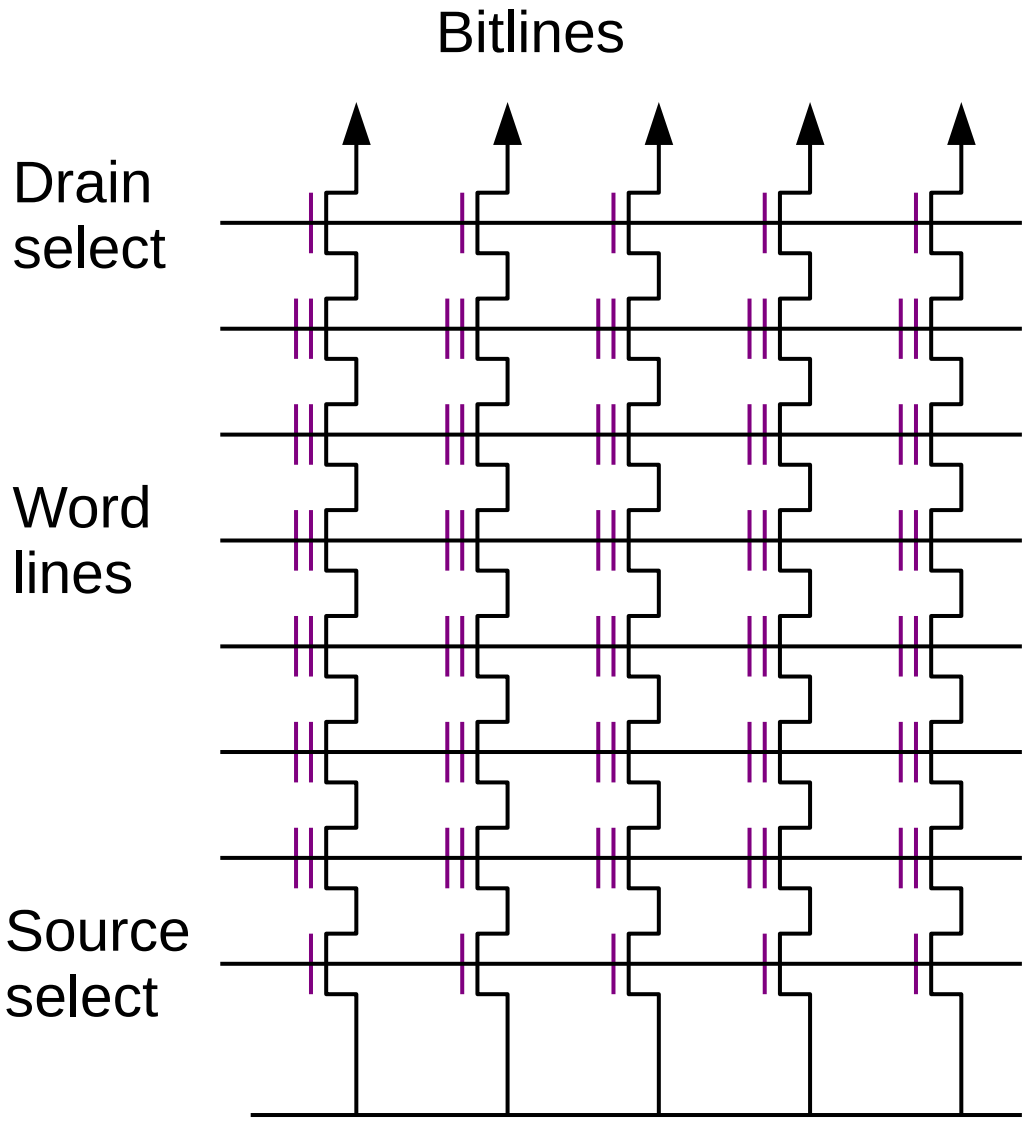
Page N and N+6
on same wordline

Upper page always
before lower page

How flash works

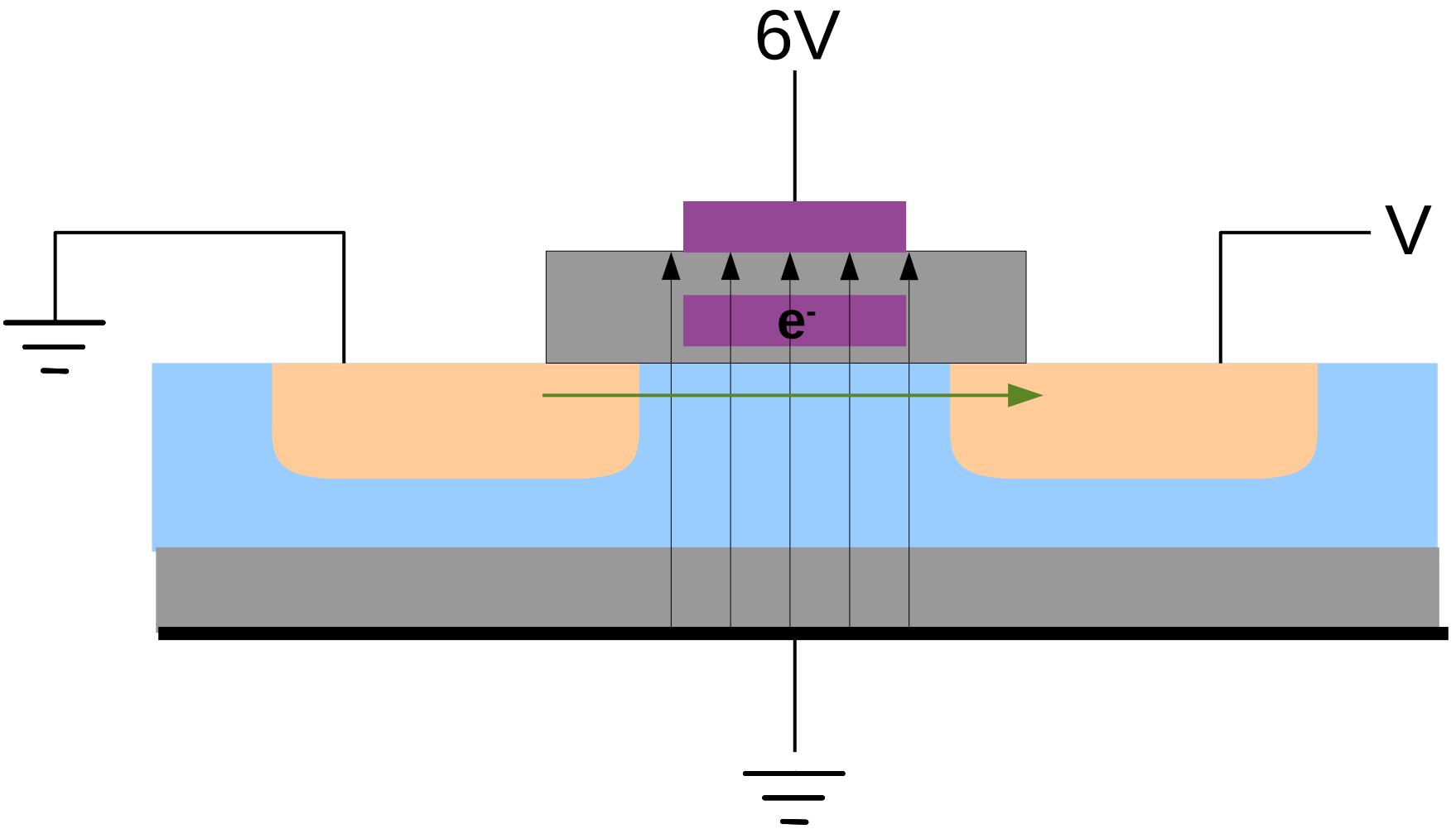
Reading

Reading a cell

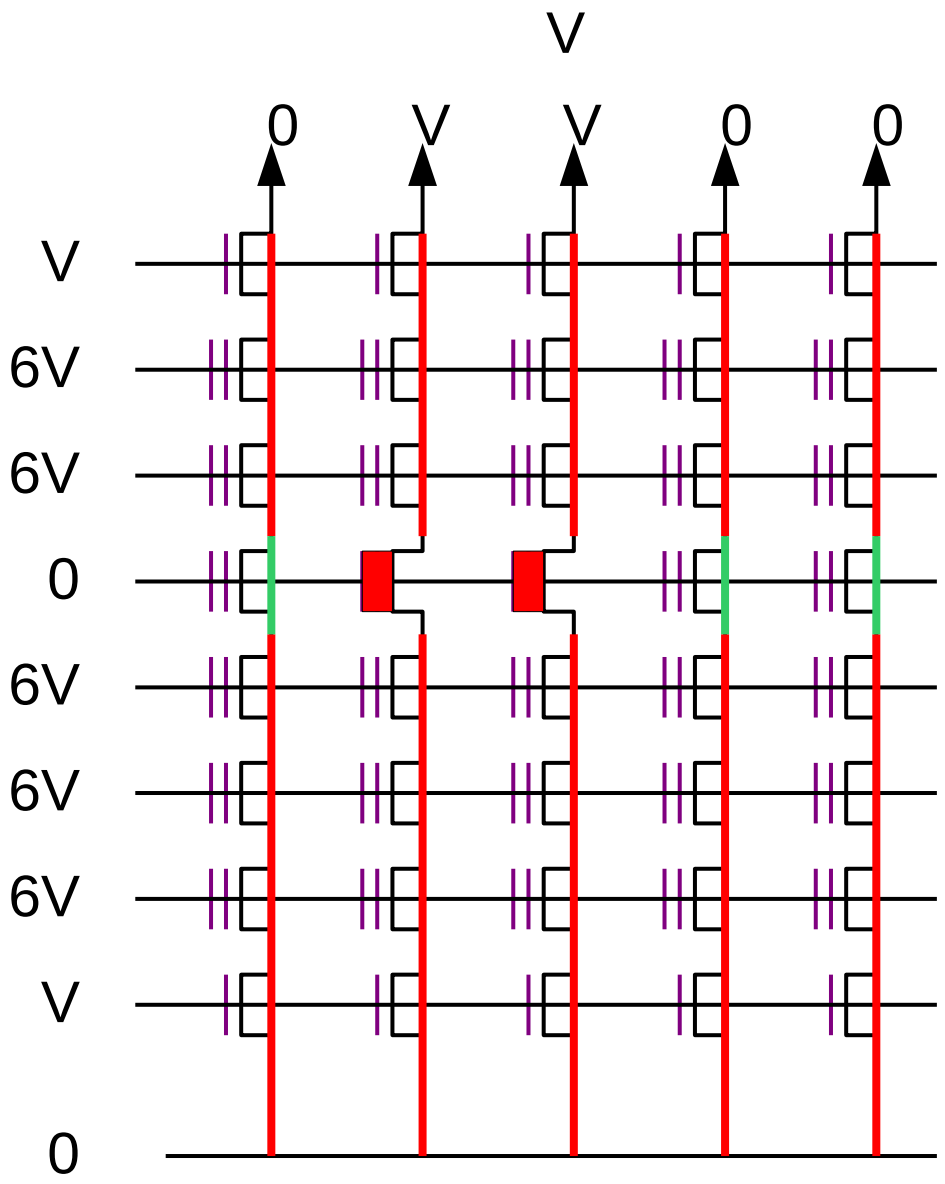


- Bitlines read in parallel
- Measure current

Bypass the floating gate with higher voltage

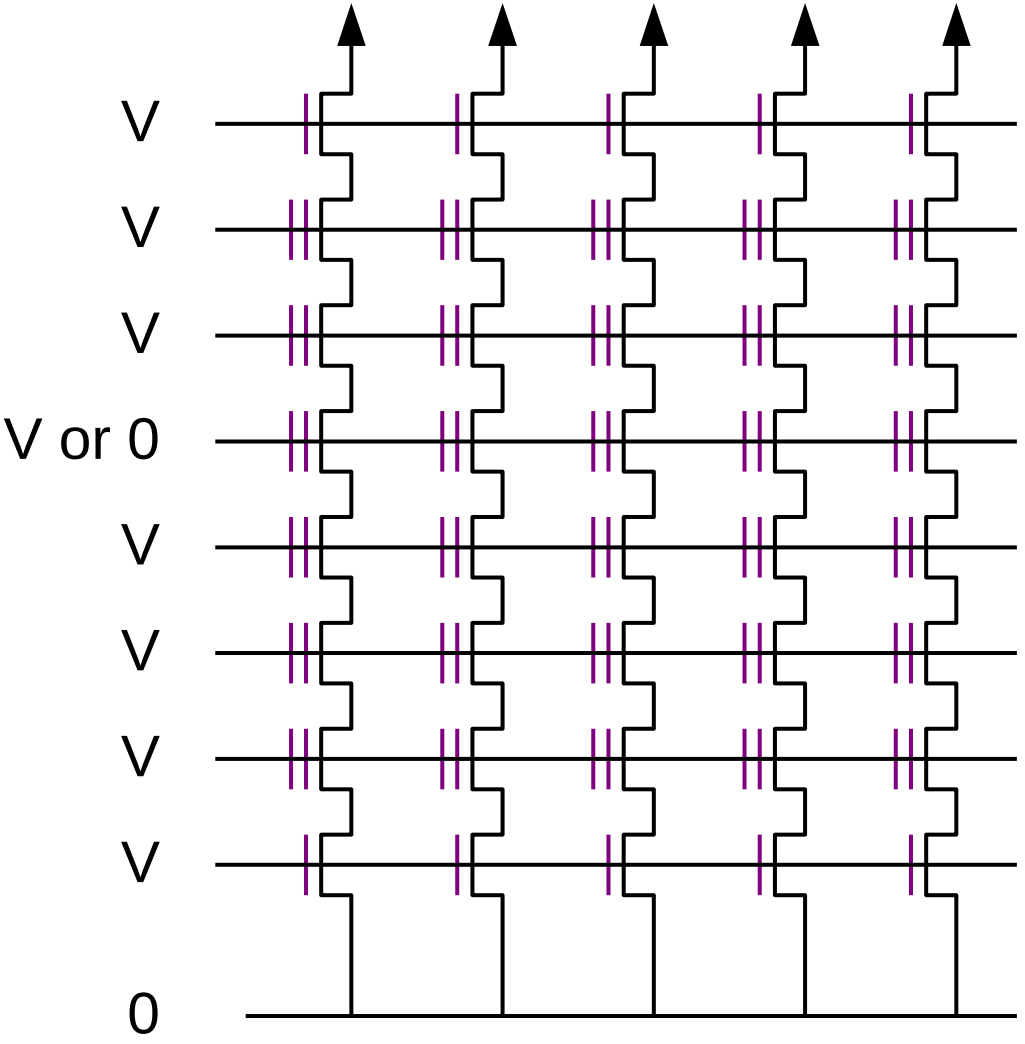


Reading a cell

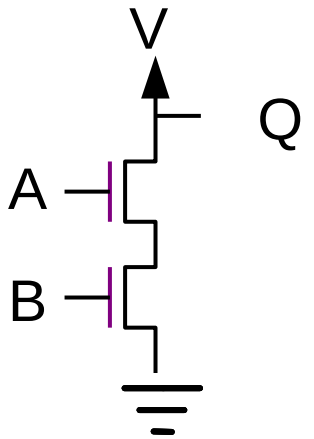


- Bitlines precharged
- Non-selected wordlines strongly enabled
- Selected wordline normally enabled
- If cell is erased (1) bitline discharges

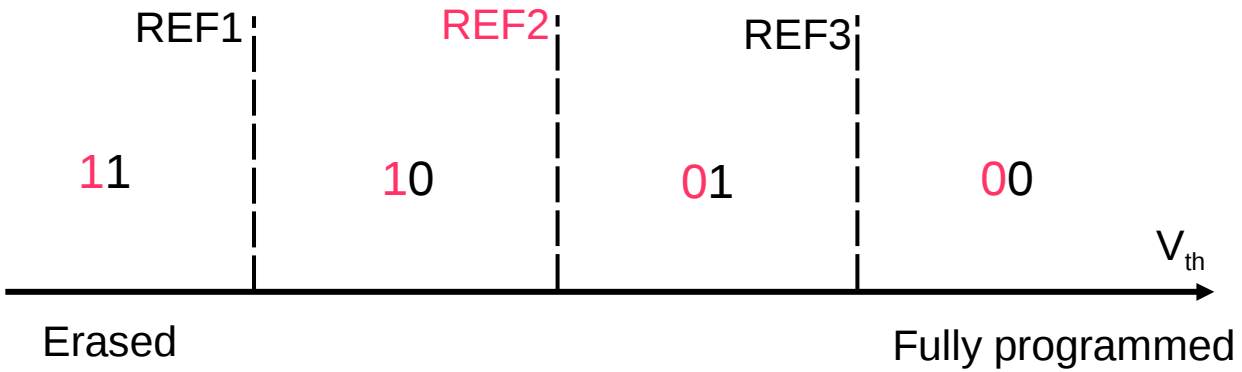
NAND structure is like CMOS NAND gate



A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

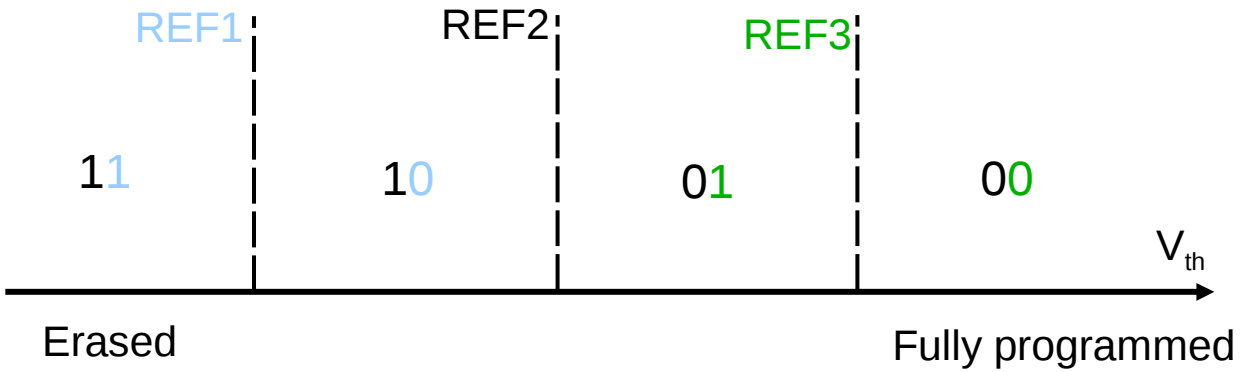


Reading a MultiLevel Cell Upper page



Read @REF2

Reading a MultiLevel Cell
Lower page requires 2 reads



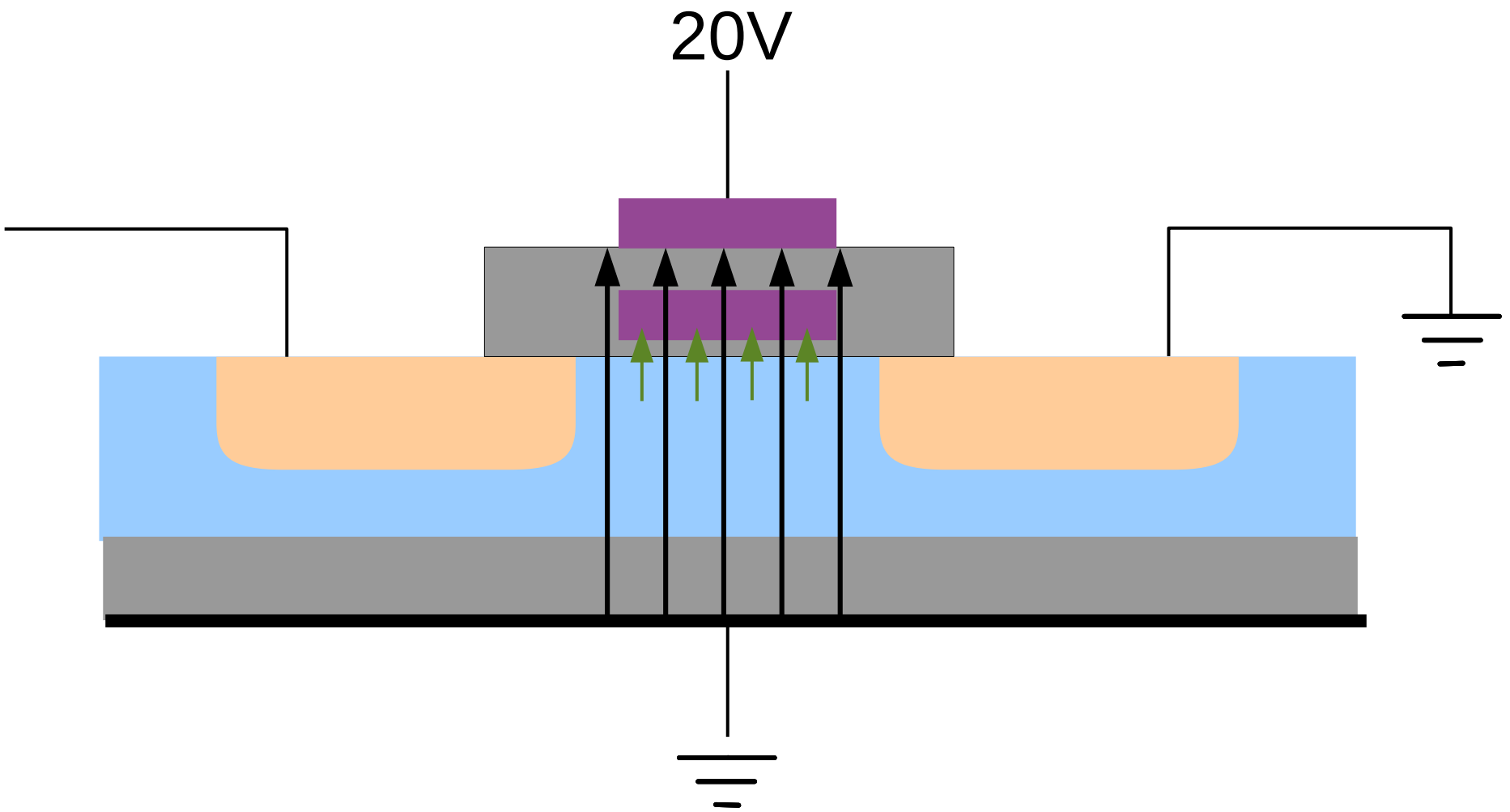
1. Read @REF3
2. Store as S1
3. Read @REF1
4. Output S1 OR S2

REF1	REF3	OUT
1	1	1
0	1	1
0	0	0

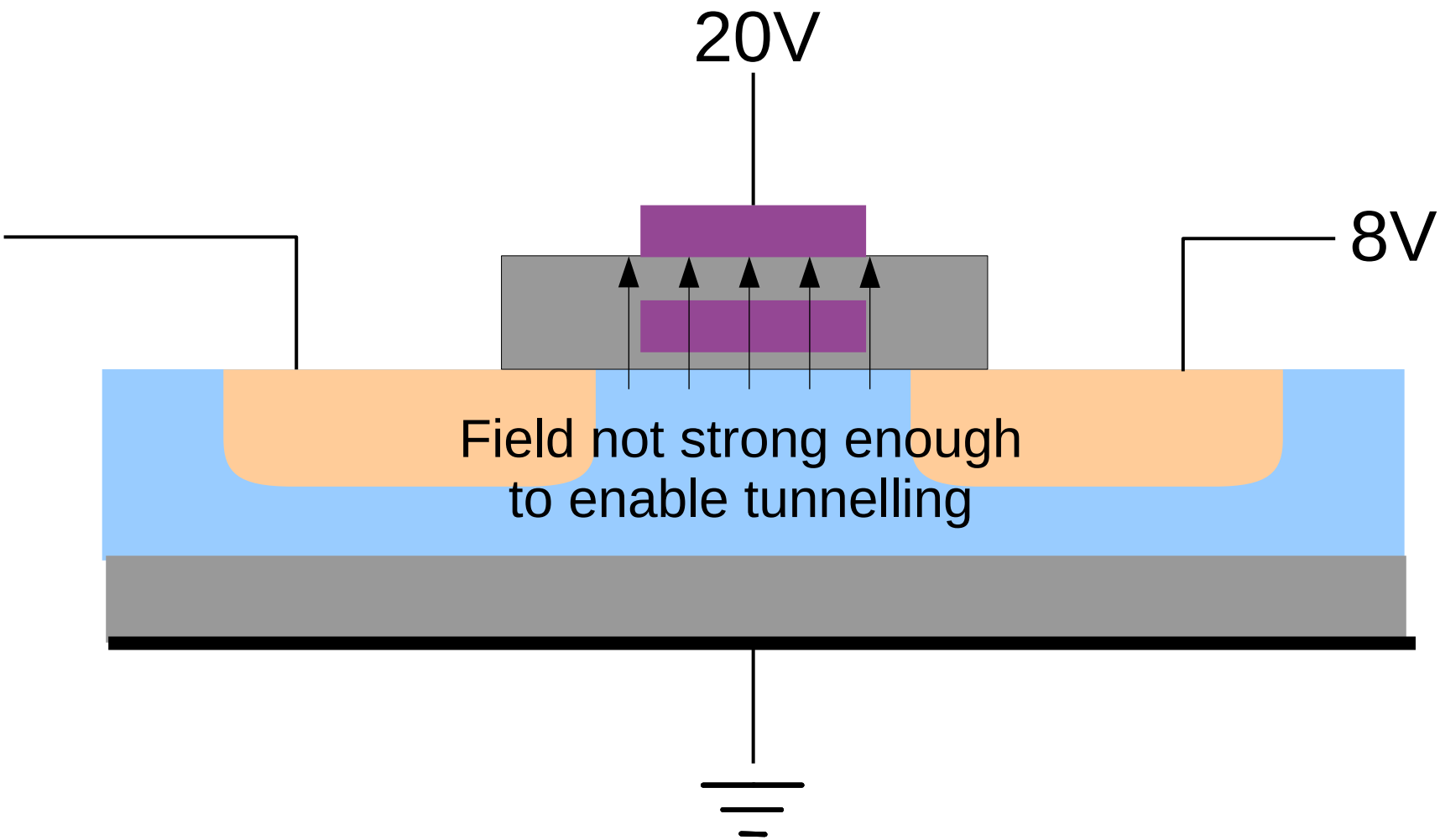
How flash works

Programming

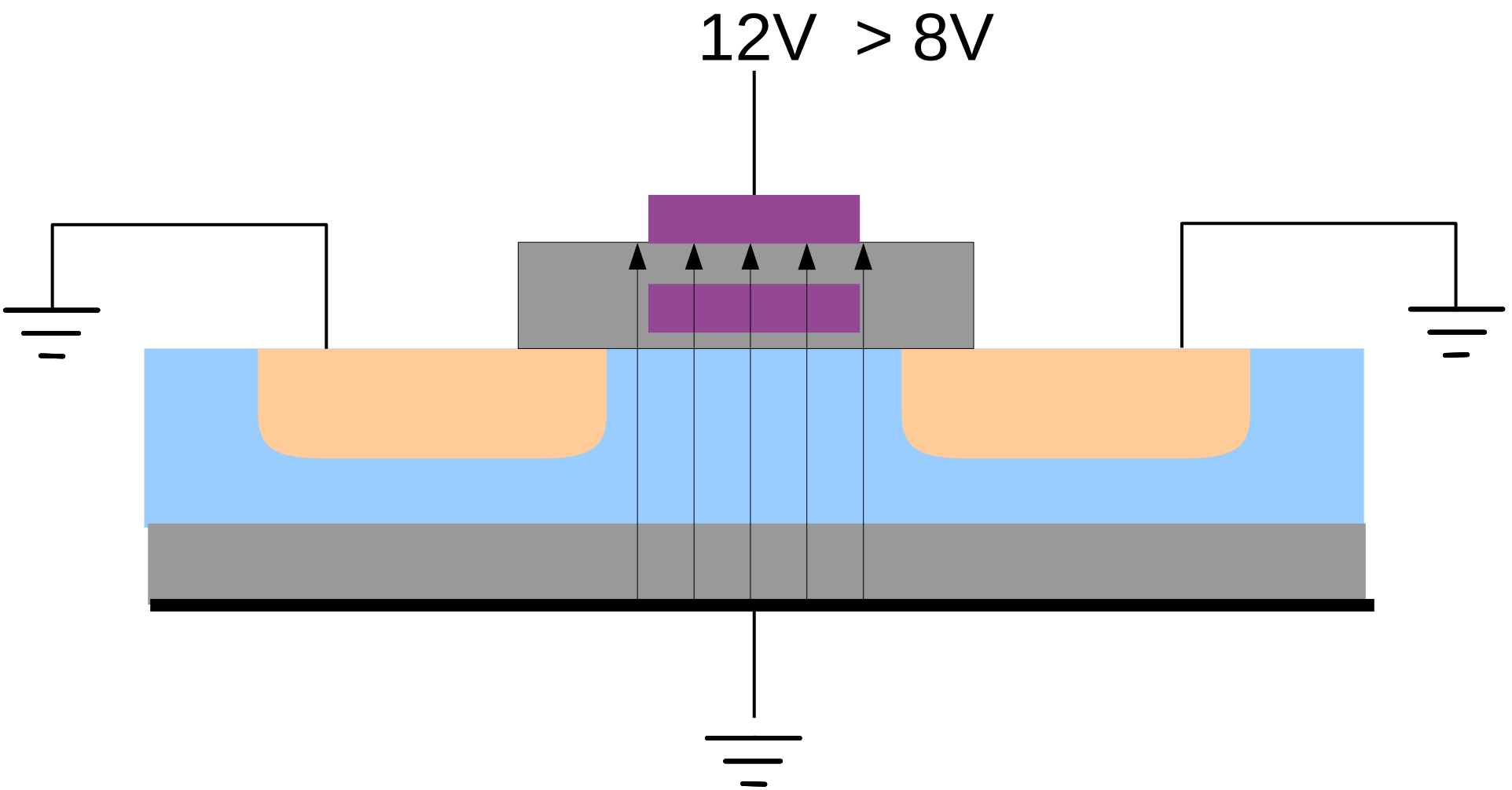
Programming the floating gate: Fowler-Nordheim tunnelling



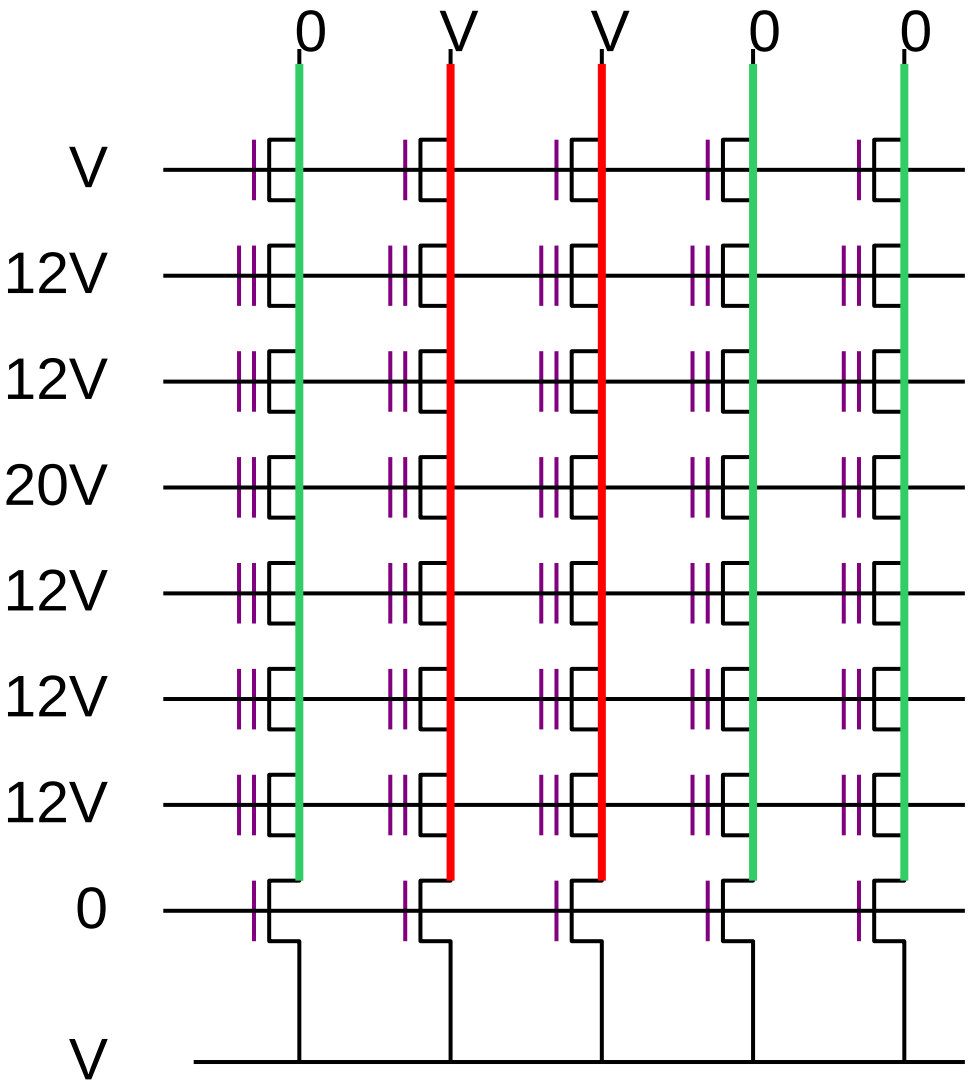
Inhibit programming to keep at 1



Bypass non-programmed wordlines

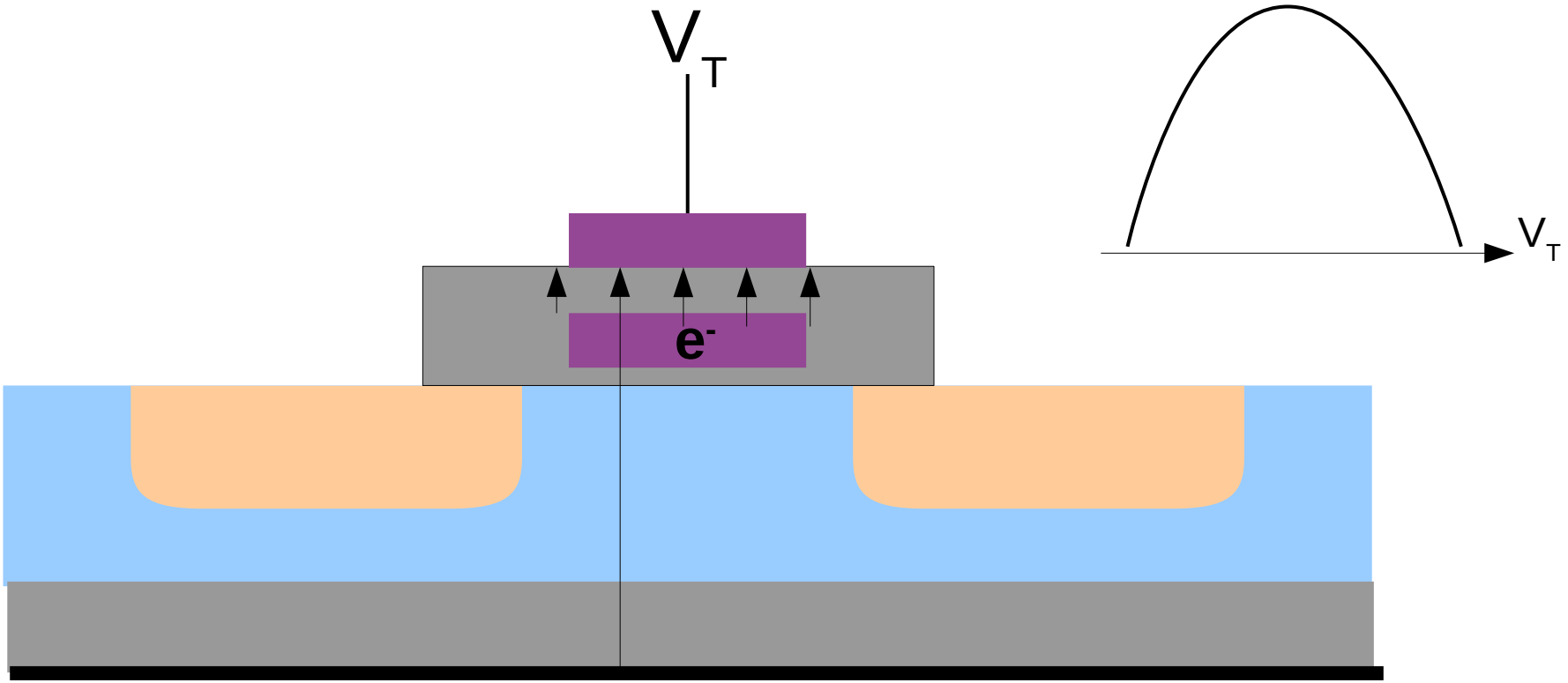


Programming a cell

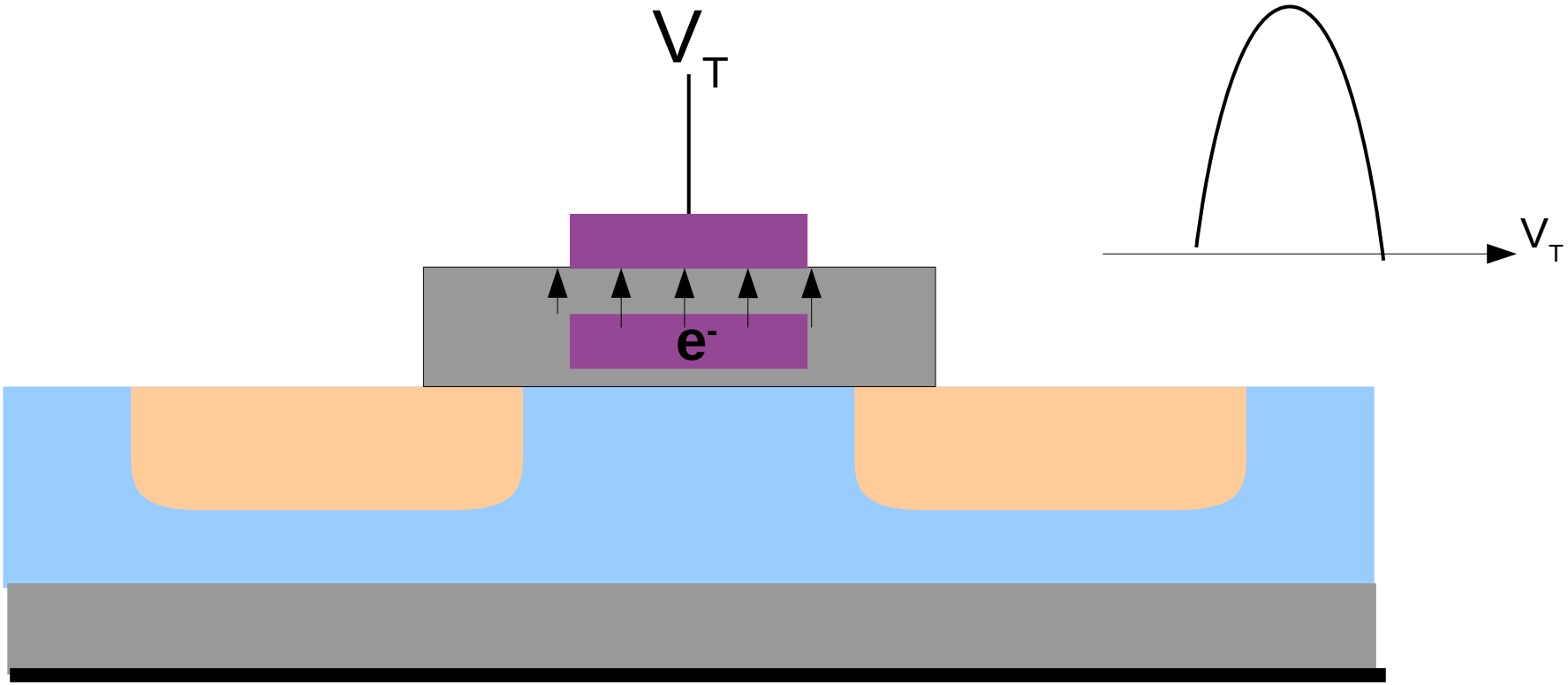


- Non-selected wordlines strongly enabled
- Selected wordline very strongly enabled

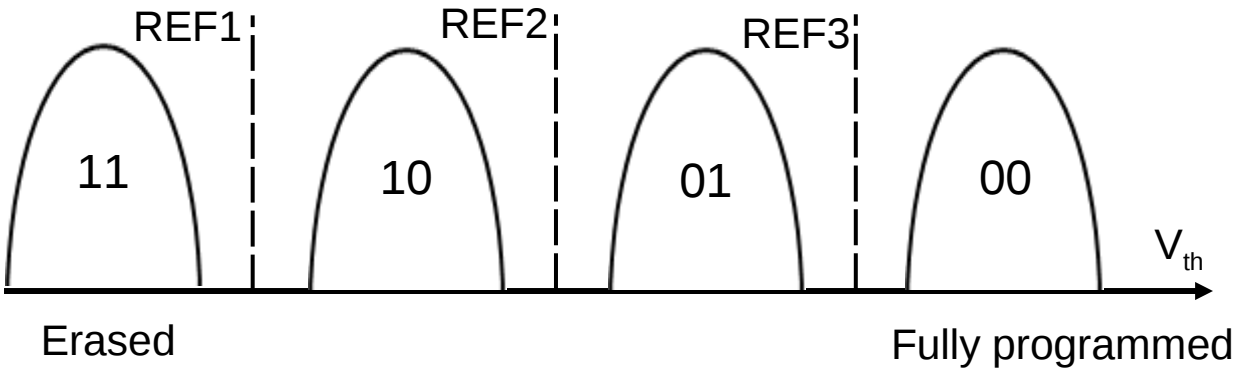
Number of electrons is stochastically distributed

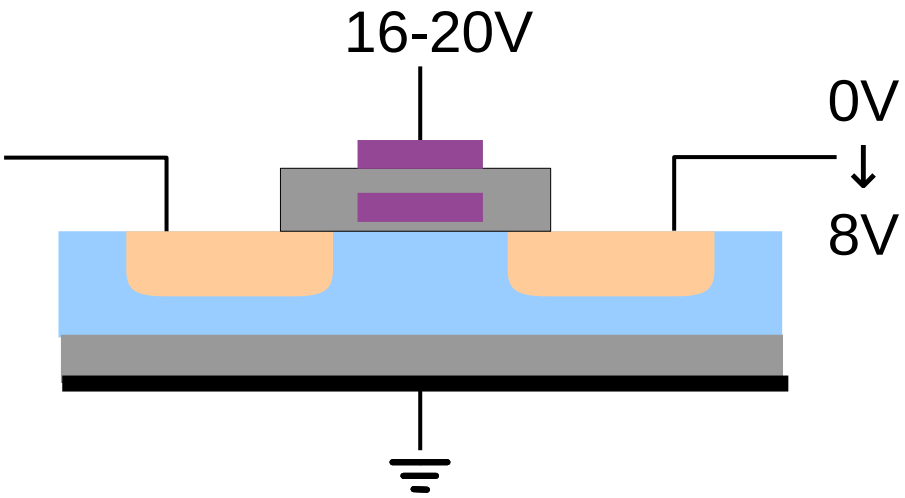


Program in pulses to narrow distribution



Program MLC by pulsing and sensing





Sense during pulse off

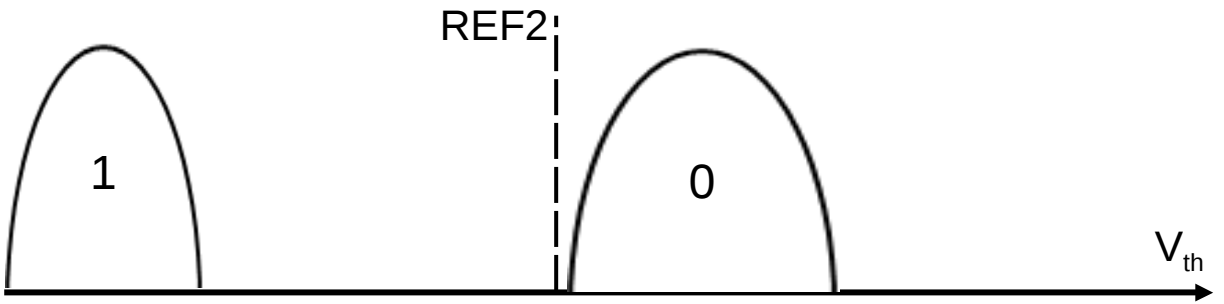
Increasing V

Stop when REF is reached
→ inhibit

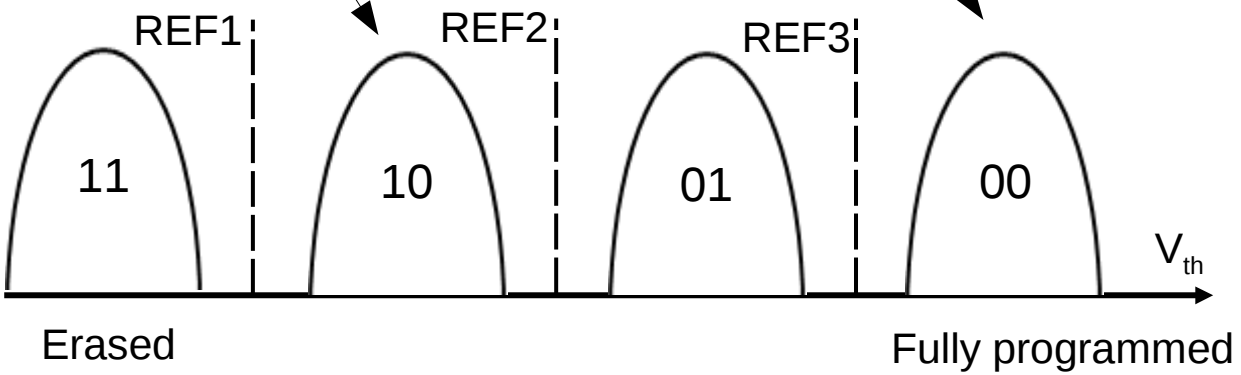
Each bitline individually

Program MLC
Upper page – Lower page

First program upper page



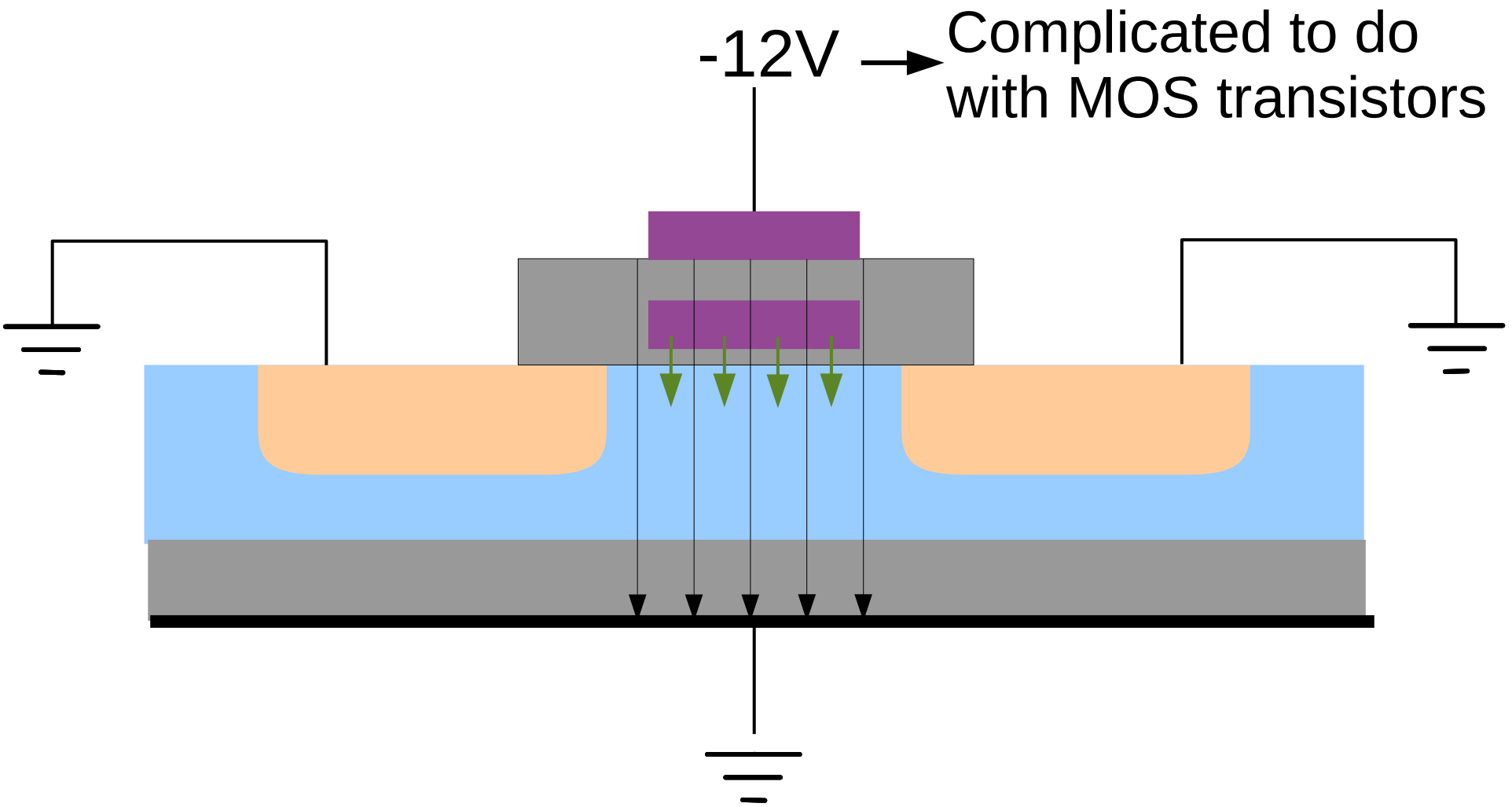
Additional program lower page



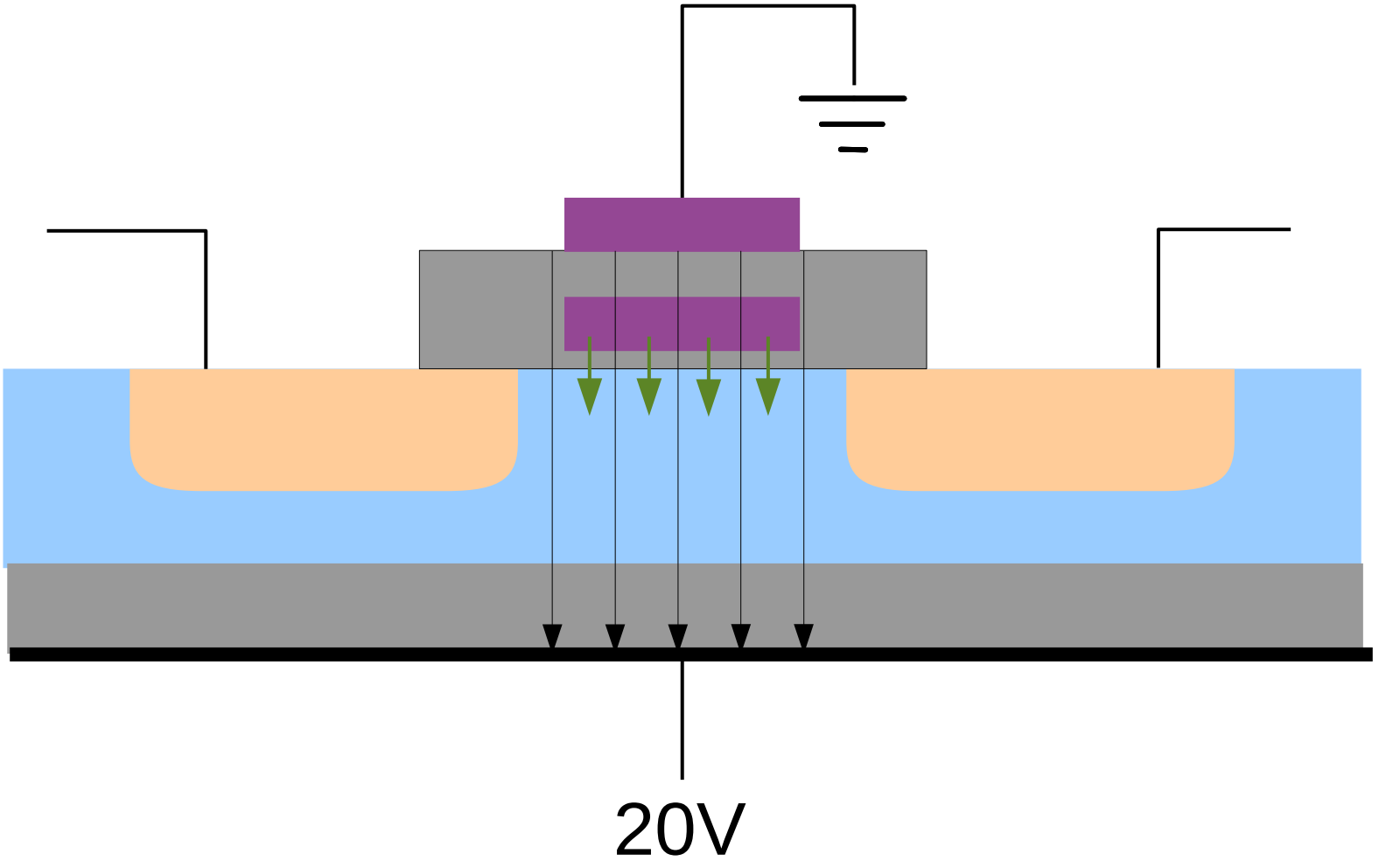
How flash works

Erasing

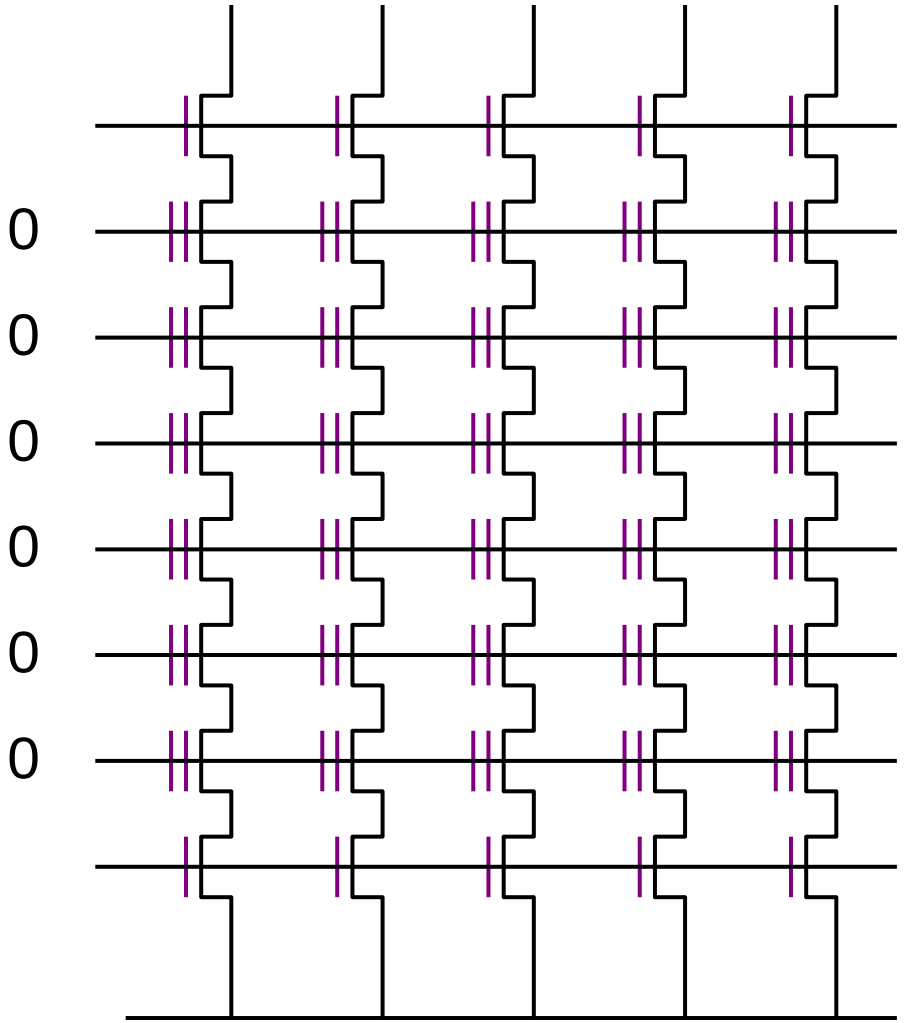
Erasing the floating gate: Reverse F-N tunnelling



Reverse the field by charging bulk

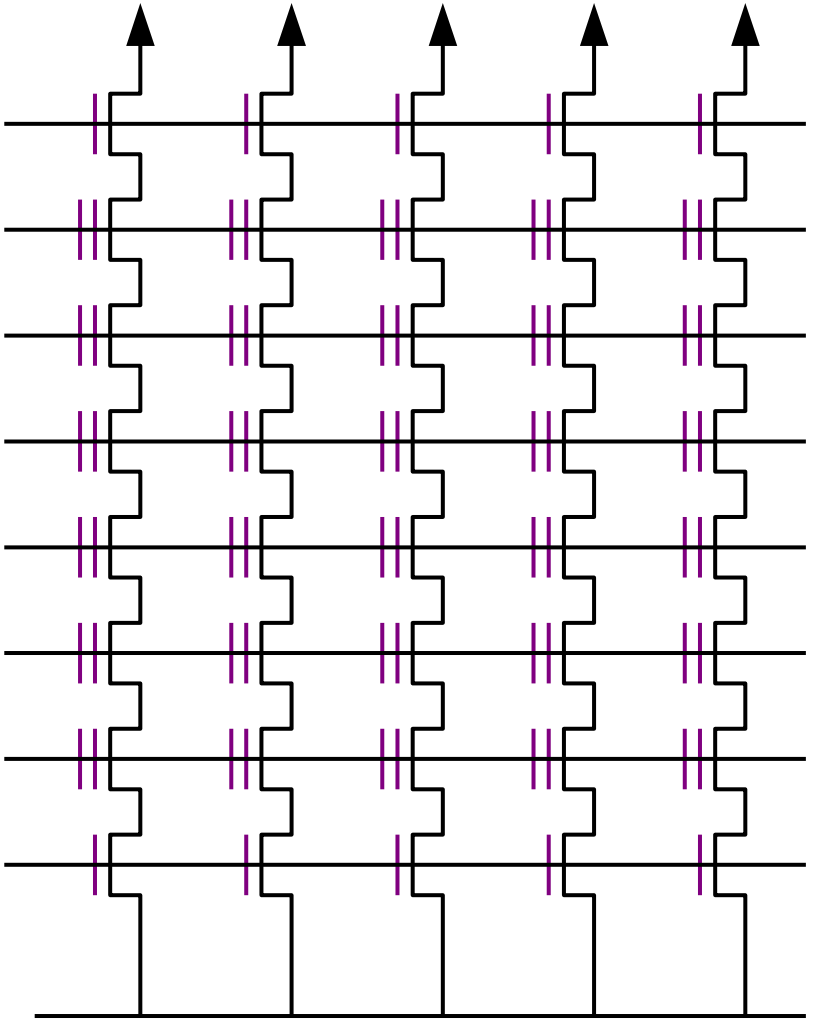


Erasing a block



- Bulk is entire block
- Select transistors can't be enabled
- They are floating

Combining cells in NAND



String of 32 cells

2*16896 strings
= 64 pages
= 1 eraseblock

How flash fails to work

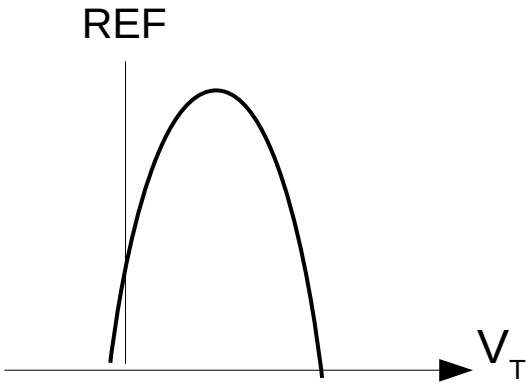
- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

How flash fails to work

- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

Bit flips

- Some cells don't reach V_{REF}
- Some cells are not fully erased
- Worse for MLC
- Work-around:
  Error Correcting Codes (ECC)
- ECC can **detect** or **correct**
  N bitflips



Error Correcting Codes

- Hamming
  - Correct 1 bit, detect 2 bit
  - 2*n parity bits protect 2n data bits

    e.g. 3 ECC bytes protect 512 data bytes
- Bose, Ray-Chaudhuri, Hocquenghem (BCH)
  - Correct M bits over 2n data bits with n*M parity bits
  - E.g. BCH16: 27 ECC bytes protect 512 data bytes
- Low Density Parity Code
  - Cfr. Hamming but more freedom of #parity bits
  - Can use soft information (i.e. how close to VT)
  - Not (yet) supported in Linux

Working with ECC

- Extra area in Flash to store ECC
  OOB = Out Of Band area
- When writing, also write ECC
- When reading, calculate ECC and compare
- Correction algorithm if calculated ECC is wrong

  syndrome = offset of the flipped bits

- **Calculating ECC requires entire page**

Hardware support for ECC

- SoCs have a hardware block to calculate ECC sometimes also calculate syndrome
- Limits the possibilities
  - E.g. TI: BCH4, BCH8, BCH16 over 512 byte chunks
  - Also limited to specific OOB locations
  - ECC may be in the middle of the page
- Some NAND chips have built-in ECC
  - Matches minimal required ECC
  - Non-standard commands to read/write with ECC
  - AFAIK not supported in Linux

Linux specifies ecc in devicetree

- nand-ecc-mode: none, soft, hw, hw_syndrome
- nand-ecc-algo: hamming or bch
- nand-ecc-strength: # corrected bits
- nand-ecc-step-size: # bytes per ECC chunk
- Not possible to specify per partition

  Annoying for first-stage bootloader which has fixed ECC

  Patch set from Boris Brezillon

How flash fails to work

- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

Access limitations

- Eraseblocks
  → Erase full eraseblock before writing
- Write once
- ECC
  → Read and write full sub-pages (512 bytes)
- MLC
  → Write full pages
  → Write upper page then lower page

Flash file system takes care of access limitations

jffs2, yaffs2, ubifs

- Only write to erased pages
- Collect writes until a page is filled
- Copy-on-write filesystem
- Journal to deal with power failure

Erased pages (= empty space) require special handling

ECC(0xFF) != 0xFF

- Each page can be written only once

  → must distinguish erased page from 0xFF
- ECC of erased page will be wrong
- Software must detect this and return 0xFF page
- Flashing tools should not write all-0xFF pages

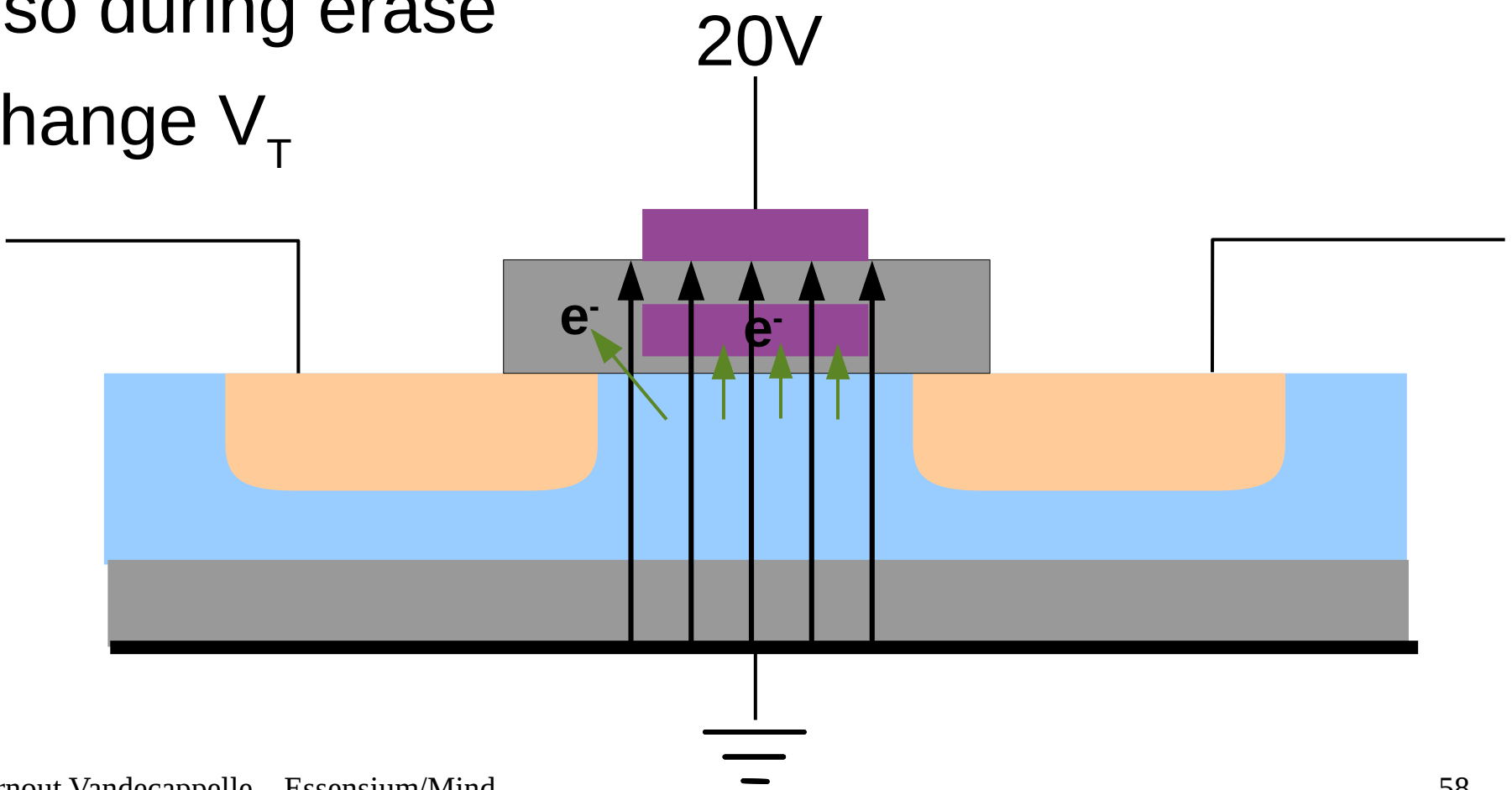
  Use `ubiformat`

How flash fails to work

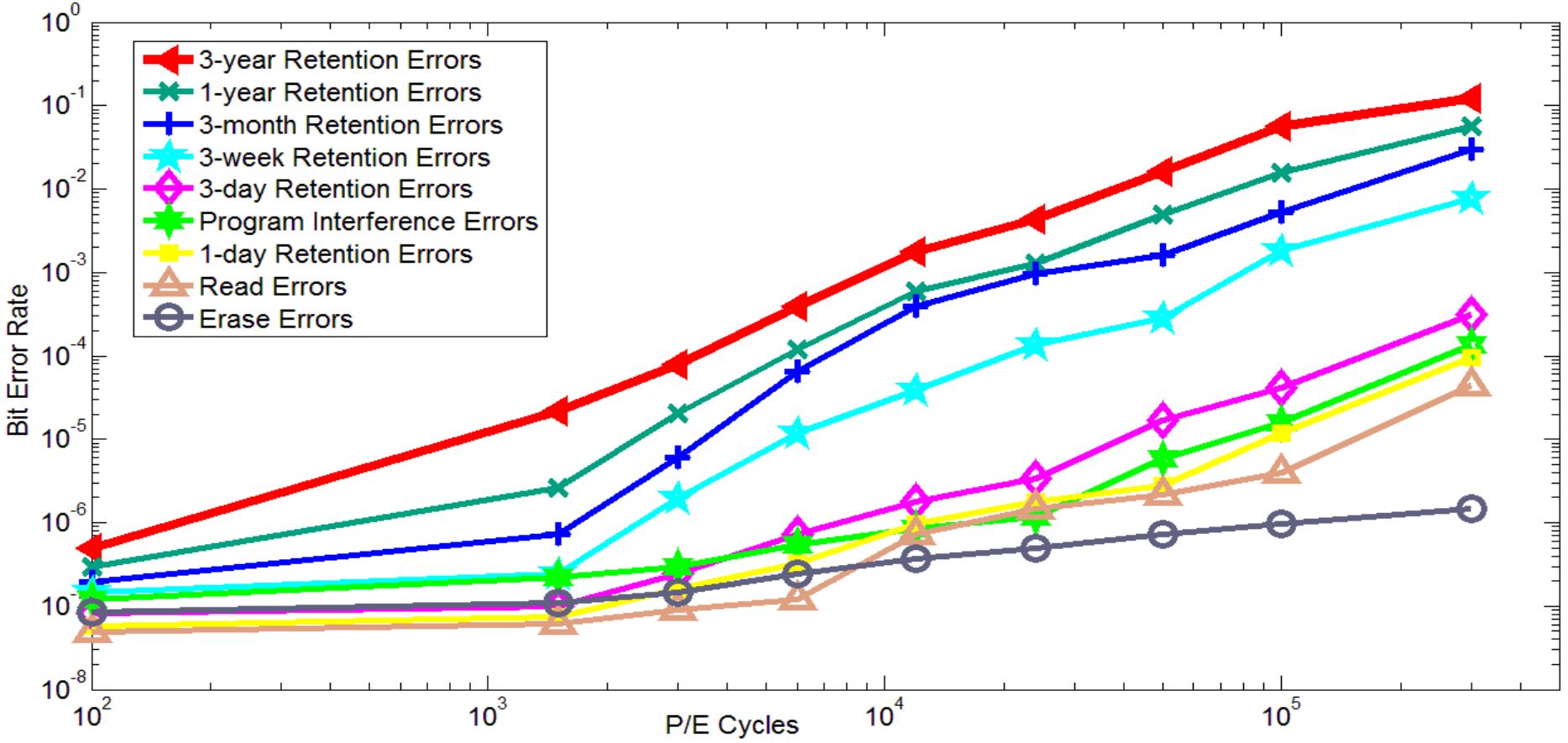
- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

Program/Erase cycles

- Some electrons captured in dielectric
- Don't escape easily
  also during erase
- Change V_T



Program/Erase cycles



Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis

Yu Cai, Erich F. Haratsch, Onur Mutlu and Ken Mai

Manage bad blocks

- When #errors is too high, mark block as bad
- Flash chip detects failed erase
- Flash chip detects write errors
- Detect when #corrected errors is high
- Torture eraseblock to confirm

  erase, check 0xff, write pattern, check pattern

  ⇒ Scrubbing
- Some blocks already marked bad in factory

Store bad blocks

- Two bytes in OOB of 1st page
  - 0xFF 0xFF ⇒ OK

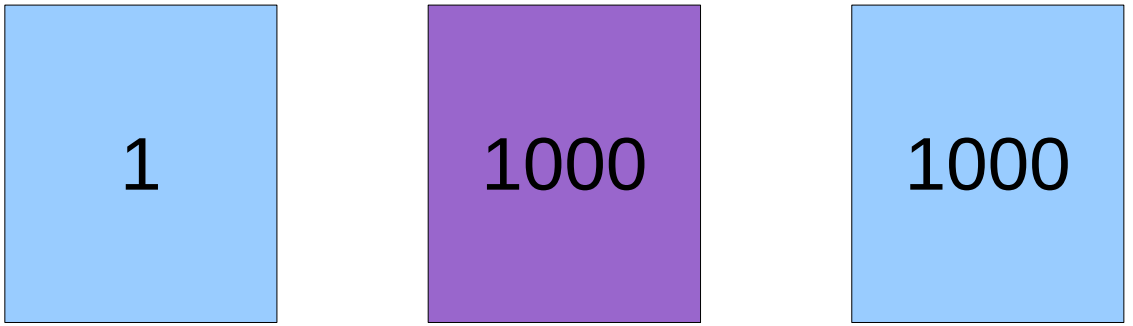
    anything else ⇒ BAD
  - Consumes 2 OOB bytes → less space for ECC
    BCH4 / 512 bytes = 8 bytes → 64 bytes / 2KB
  - Used by factory-marked bad blocks

- Bad block table at well-known flash location
  - Must have enough spare, if BBT itself goes bad
  - Must have 2 copies to deal with power failure

Limit Program/Erase cycles = Wear levelling

- Don't write to same location all the time
- Keep track of #erase
- Write to block with lowest #erase

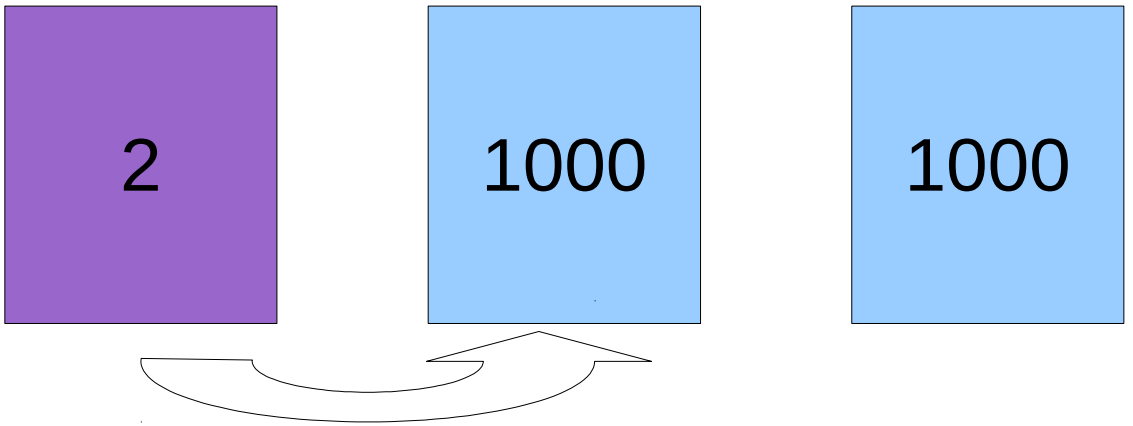
  "write head"
- Migrate data when #erase is inbalanced



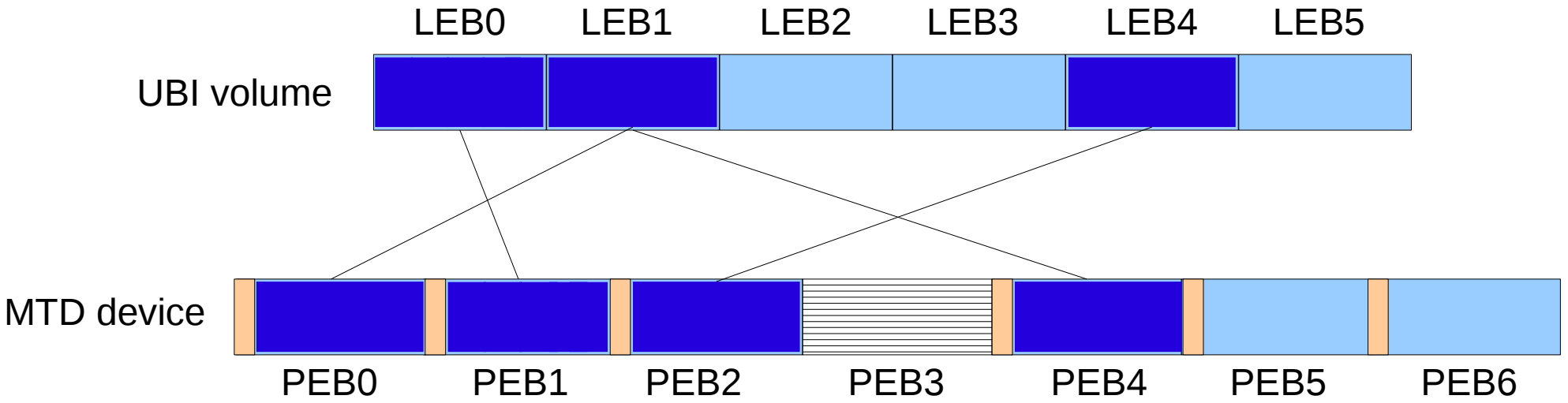
Limit Program/Erase cycles = Wear levelling

- Don't write to same location all the time
- Keep track of #erase
- Write to block with lowest #erase

  "write head"
- Migrate data when #erase is inbalanced



UBI layer manages wear leveling and bad blocks



- Header with erase counter, volume ID
- EC is written immediately after erase
- Migrate if EC difference is too high
- Several PEB may map same LEB, use latest

Bitflips in erased pages

- A cell may not have been fully erased

  ⇒ Erased block not entirely 0xFF

- Currently considered fatal by UBI

  ⇒ block is marked as bad

- IMHO too aggressive for high MLC

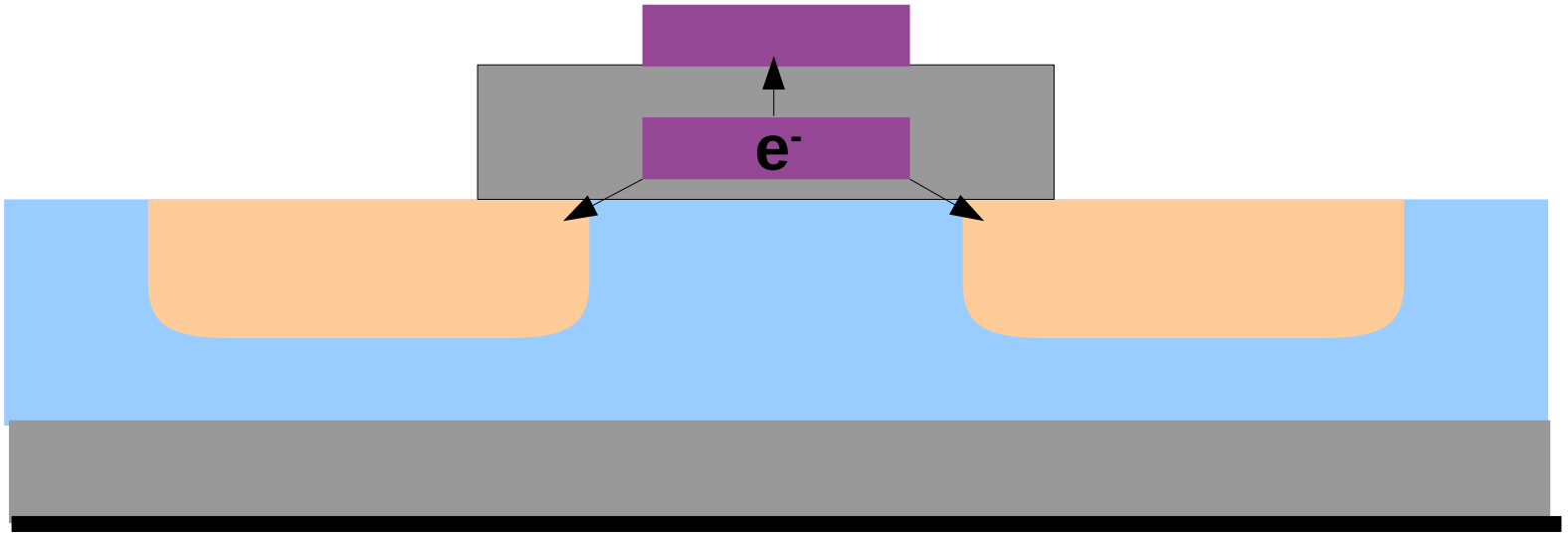
  8-bit ECC can easily protect 1 bitflip

How flash fails to work

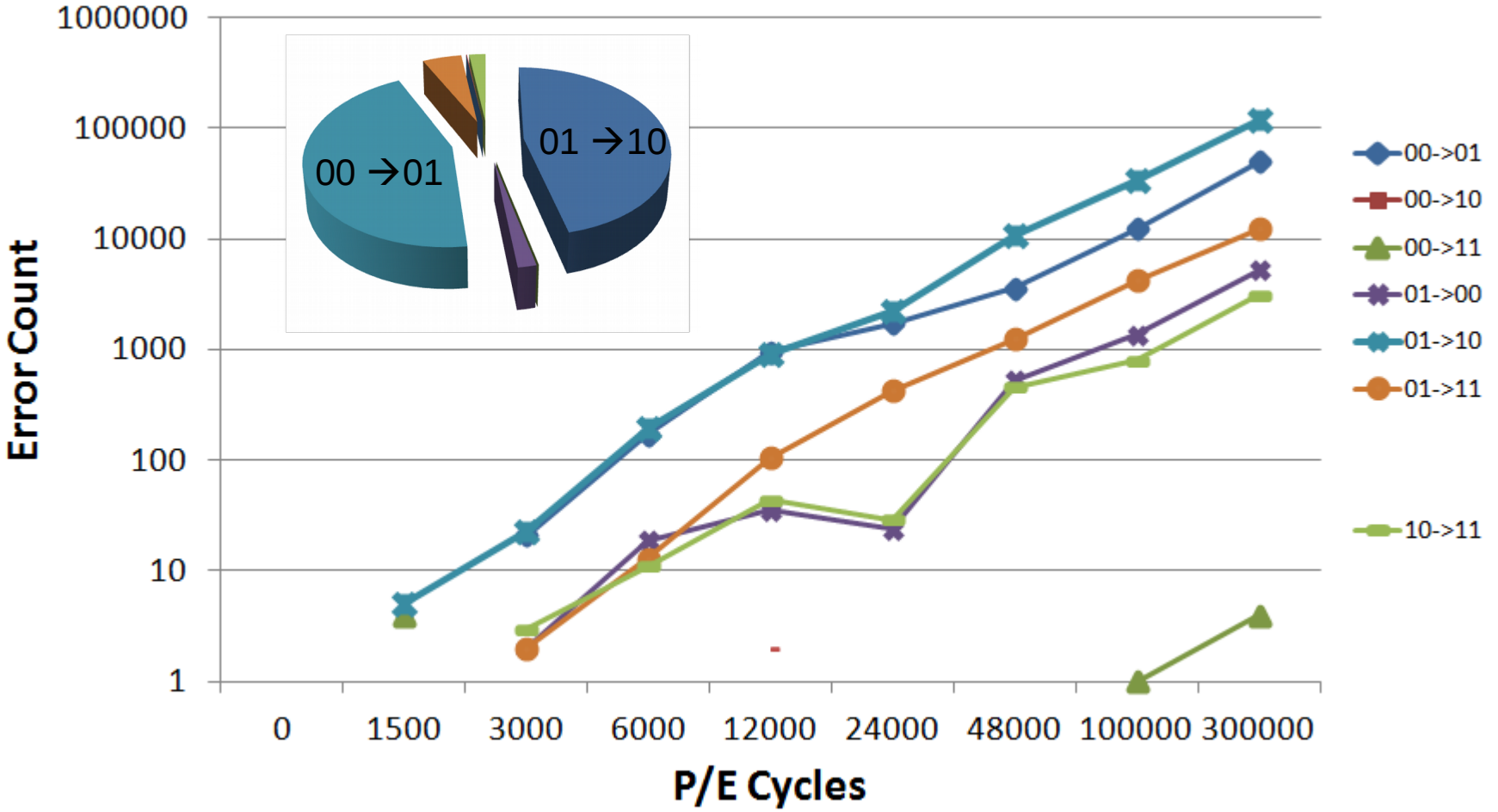
- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

Electrons leak after some time

Floating gate surrounded by insulator
but this is not perfect

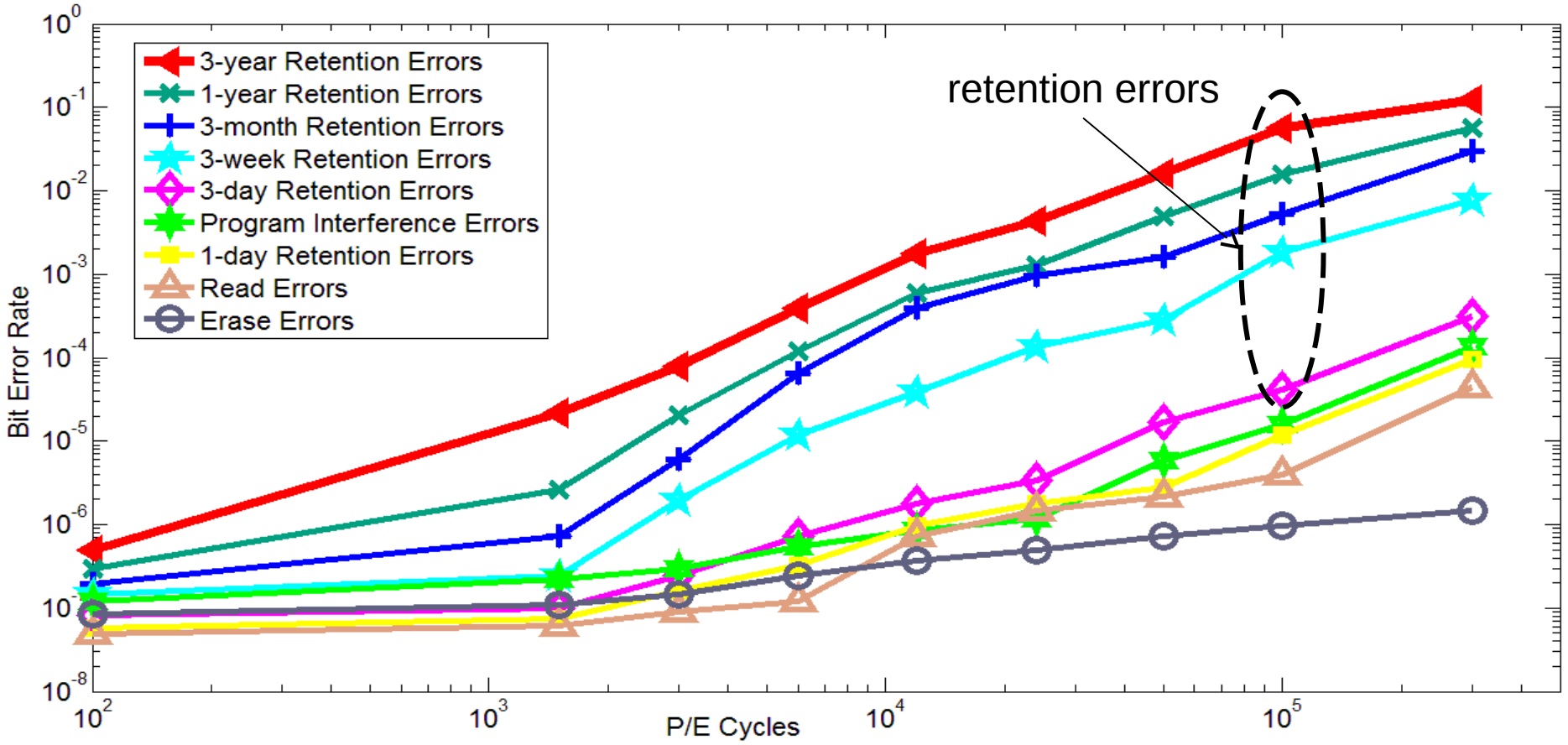


Electrons leak after some time



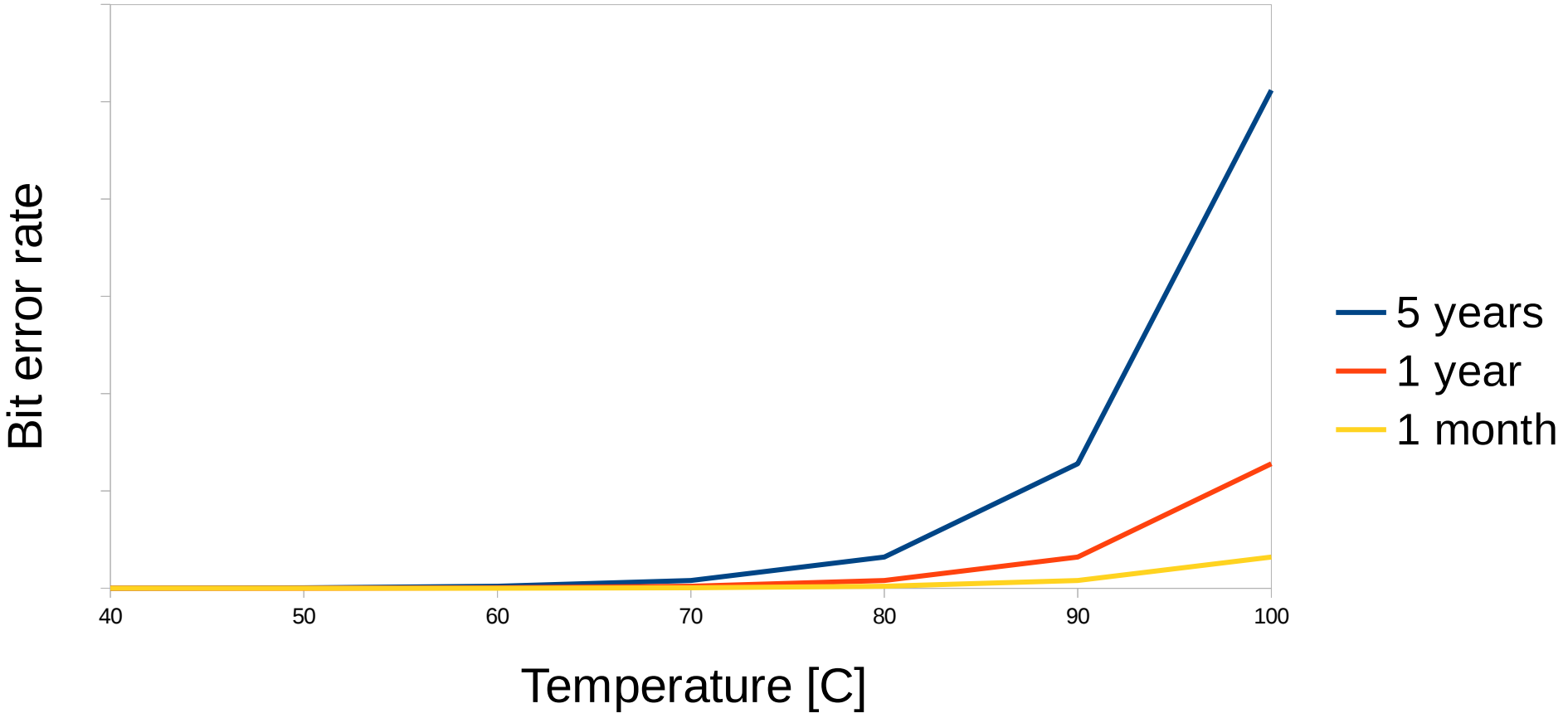
Error Patterns in MLC NAND Flash Memory

Retention is the limit of program/erase cycles



- Looks OK after writing
  but will fail after days/weeks

Retention strongly depends on temperature

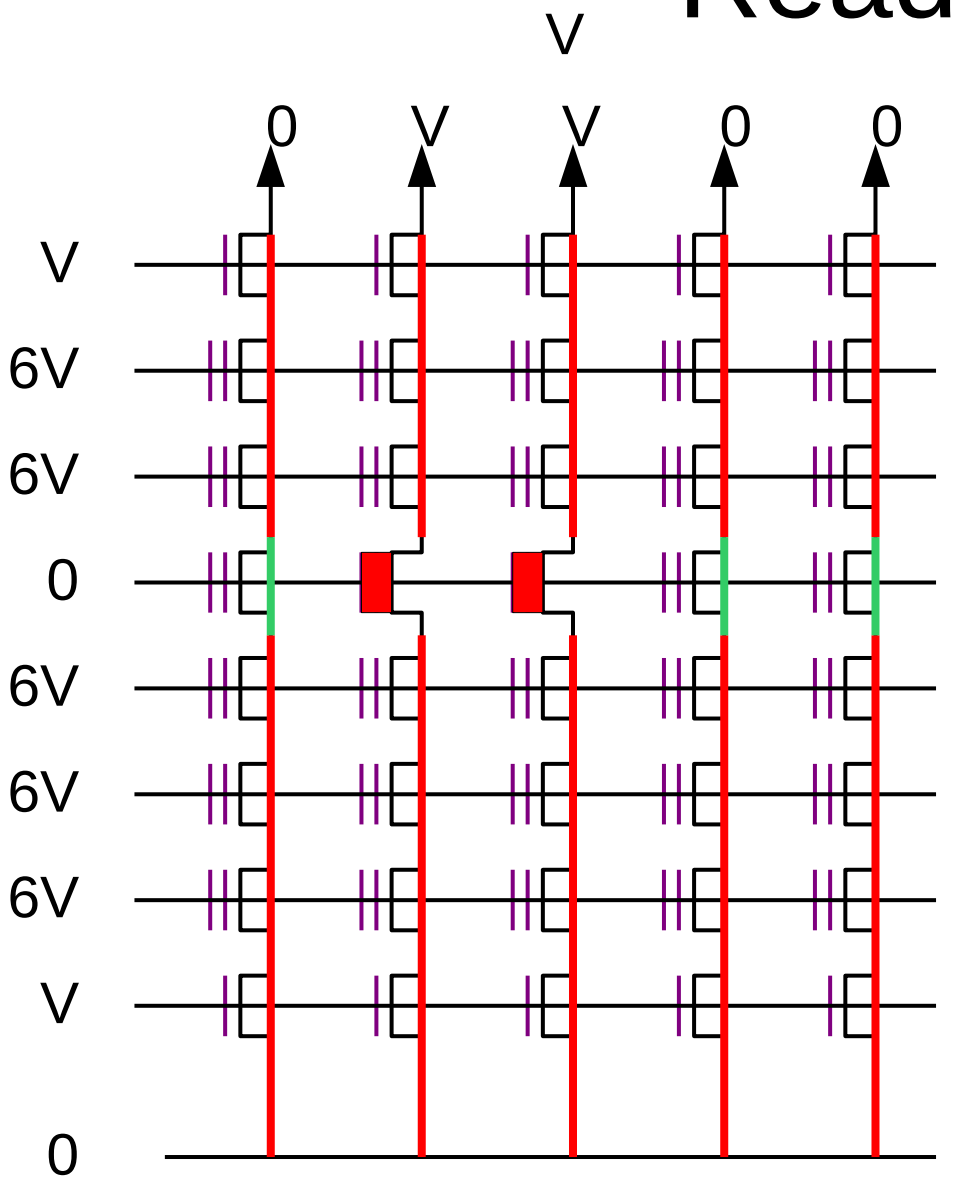


No quantitative results found

How flash fails to work

- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

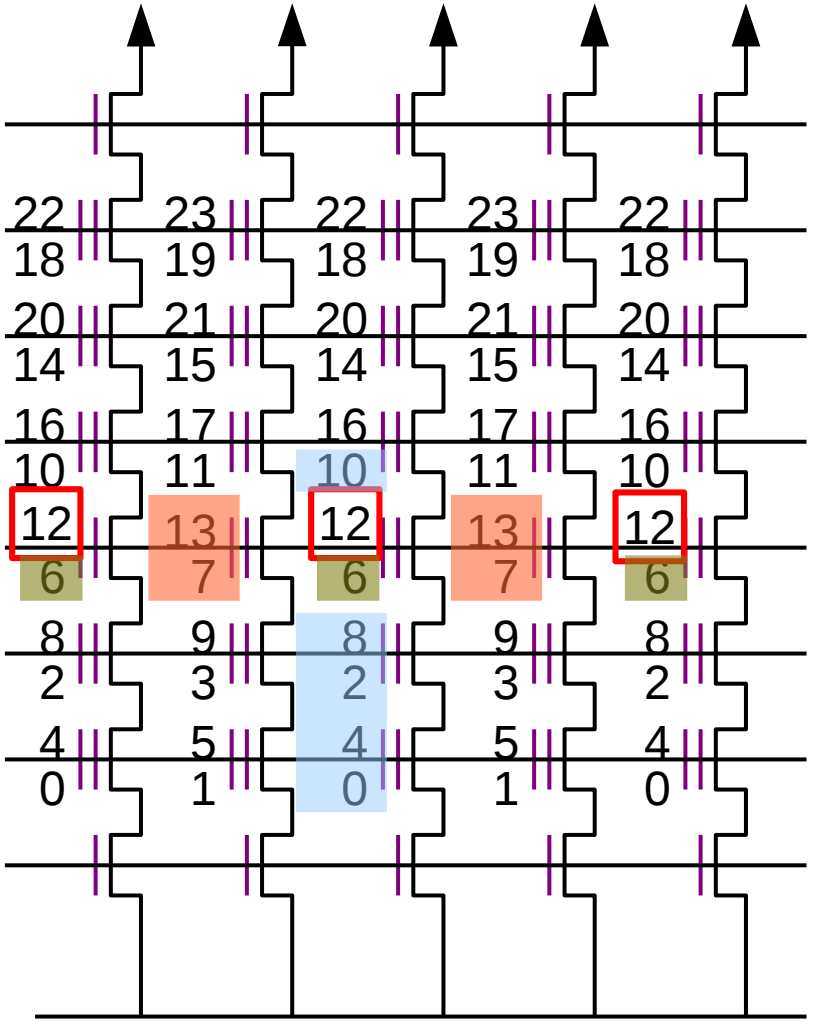
Read disturb



- Bypassed cells subject to strong field (6V)
- Risk of floating gate being affected esp. in MLC

→ When reading, bits on **other** pages may flip

Program disturb / interference



- Pages on same bitline
- Pages on same wordline
- Upper pages in same cell

- More important than read disturb but write only once
- 1→0 more likely

How to deal with disturb

- Detect bitflips and mark blocks for refresh
- Make sure that all pages are regularly read
- Also good for retention
  (as long as device stays on)
- I'm not sure how to deal with program disturb

How flash fails to work

- Bit flips
- Access limitations
- Program/Erase cycles
- Retention
- Read/Program disturb
- Power failure

When power goes out...

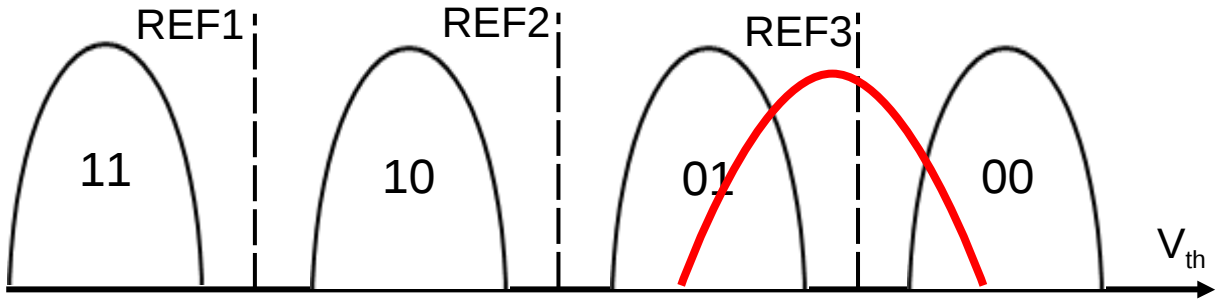
With buffered metadata

- E.g. new version is written, old one not erased yet
- Flash filesystem should take care of this
- UBIFS: sequence number on each node

When power goes out...

In the middle of a write

- Target V_{REF} not yet reached



- Some bits are 1 instead of 0
- Bits just at V_{REF}: extremely weak retention
  - Can't be detected immediately
  - Very vulnerable to read/program disturb
  - Unstable bit problem

When power goes out...

In the middle of an erase

- Not all bits below lowest V_{REF}
- Erased block not all 0xFF, but data not valid either

How to deal with unstable bits

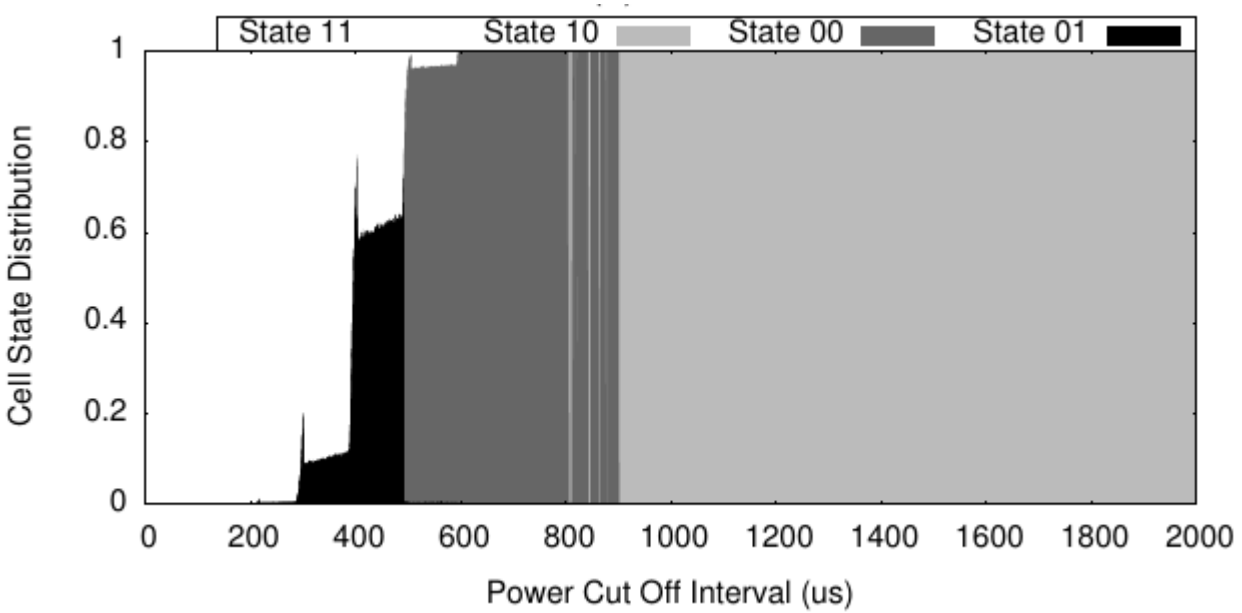
- When attaching, find LEBs potentially written
- Read once and write back to fresh PEB
- IIUC this is not fully implemented

When the power goes out...

In the middle of a write to a lower MLC page

→ Cell can be in any state

Upper bit = 1
Program lower bit
1 → 0



Understanding the Impact of Power Loss on Flash Memory

Hung-Wei Tseng, Laura M. Grupp, and Steven Swanson, DAC 2011

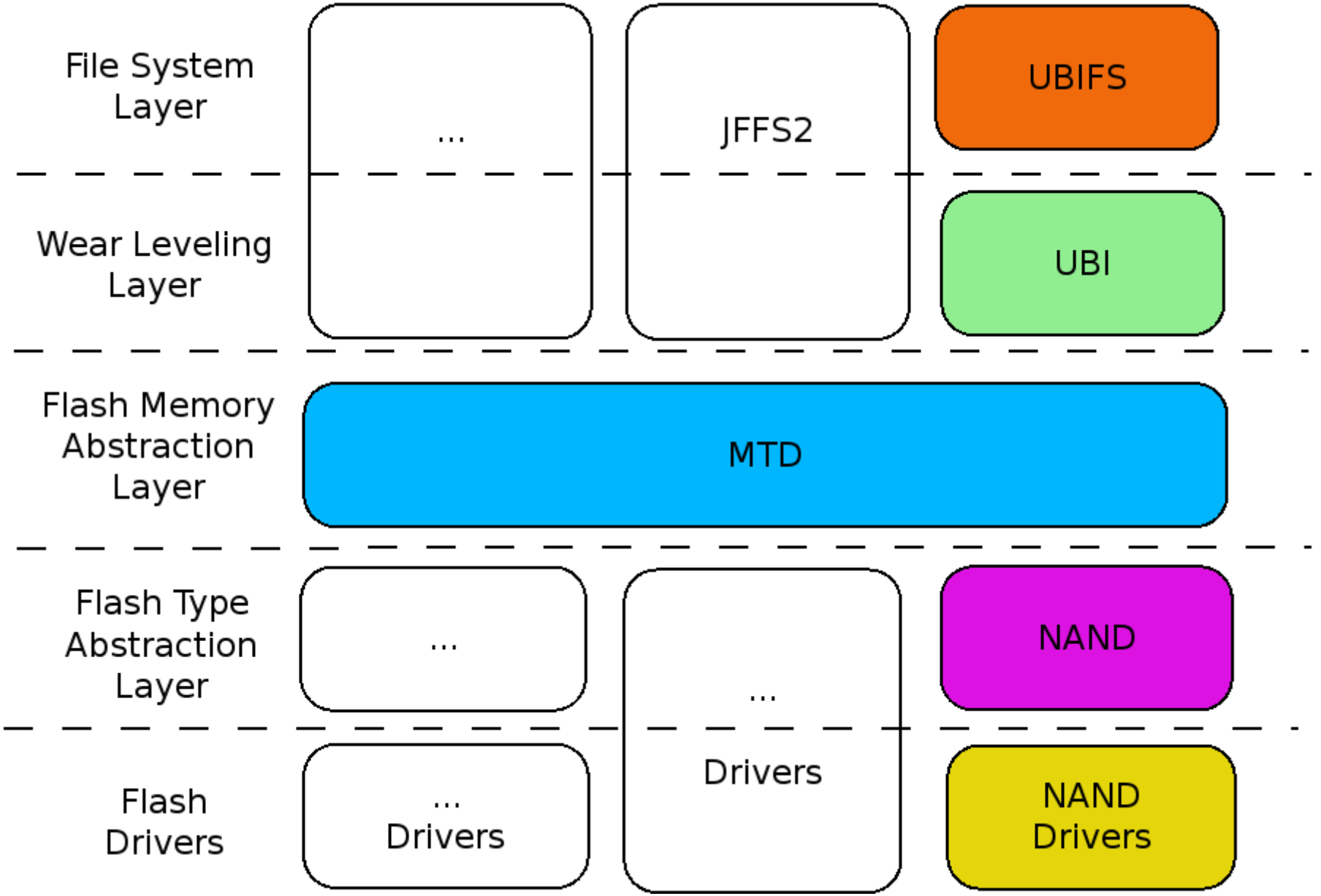
When the power goes out...

In the middle of a write to a lower MLC page

→ Cell can be in any state

- Don't use lower pages at all
- Only use lower pages when upper pages have CRC'ed data

Workarounds to make flash work



NAND driver

- Offer Erase, Write and Read primitives
- ECC handling
  Return -EUCLEAN for correctable error
  Return -EIO for uncorrectable error
- Return -EIO for failed write/erase
- Bad block handling
  - BBT or in OOB
  - Only store bad blocks, upper layer creates them
- *Currently no primitive for empty space*

UBI translation layer

- Wear levelling
  - Choose less used PEBs for writing
  - Migrate cold PEBs
  - Across *entire* flash

    Do not partition flash
- Manage multiple volumes cfr. partitions
- Bad block creation

UBI translation layer future functionality

- Block refresh

  Background reading to deal with

  - retention
  - read/program disturb
- Paired page handling
  - Distinguish safe and unsafe LEBs
  - Use unsafe LEBs e.g. for migrated PEBs

UBIFS layer

- Copy-on-write of user data
- Copy-on-write of metadata: inodes, tree
- Collect writes in a write buffer

  → write complete pages
- Recover from power cuts

Managed flash
Flash Translation Layer (FTL)

SD card, SSD, eMMC, ...

- ECC handling
- Bad block handling and scrubbing
- Wear levelling
- Write once: block- and page remapping
- Usually ***not*** powercut safe

Problems with managed flash

- No help from filesystem structure
- Depending on history
  access can take 10-100x longer
- End of life (wear, retention) usually fatal
- FTL controller is unknown
  - No specifications from manufacturer
  - Hard to test: which access pattern is worst case?
  - 2 "identical" devices not always same firmware

Optimising for managed flash

- Avoid small updates

  → write buffering
- ERASE/TRIM command for unused blocks
- f2fs (flash-friendly filesystem)

  Mostly optimise for access speed
  but this is also good for wear
- ext4 and btrfs take ideas

The future is in managed flash

- It's cheaper
- MLC hacks make workarounds more complex
  - E.g. soft info for LDPC
  - E.g. store some metadata in NOR or SLC
- Takes away CPU power
- Easier to give eMMC a dying gasp

⇒ IMHO everyone will move to eMMC

My advice

- Small capacity (<= 512MB)
  - SLC NAND with dying gasp cap
  - 1 big partition with UBI + several volumes
  - Adapt ECC strength and EC imbalance to datasheet
- Large capacity (> 2GB)
  - eMMC with dying gasp cap
  - ext4 or btrfs or f2fs

Managed NAND Flash hides complexity

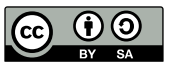
- SATA interface = SSD disk
- SD interface = eMMC

But controllers are unpredictable
Still need optimized access patterns

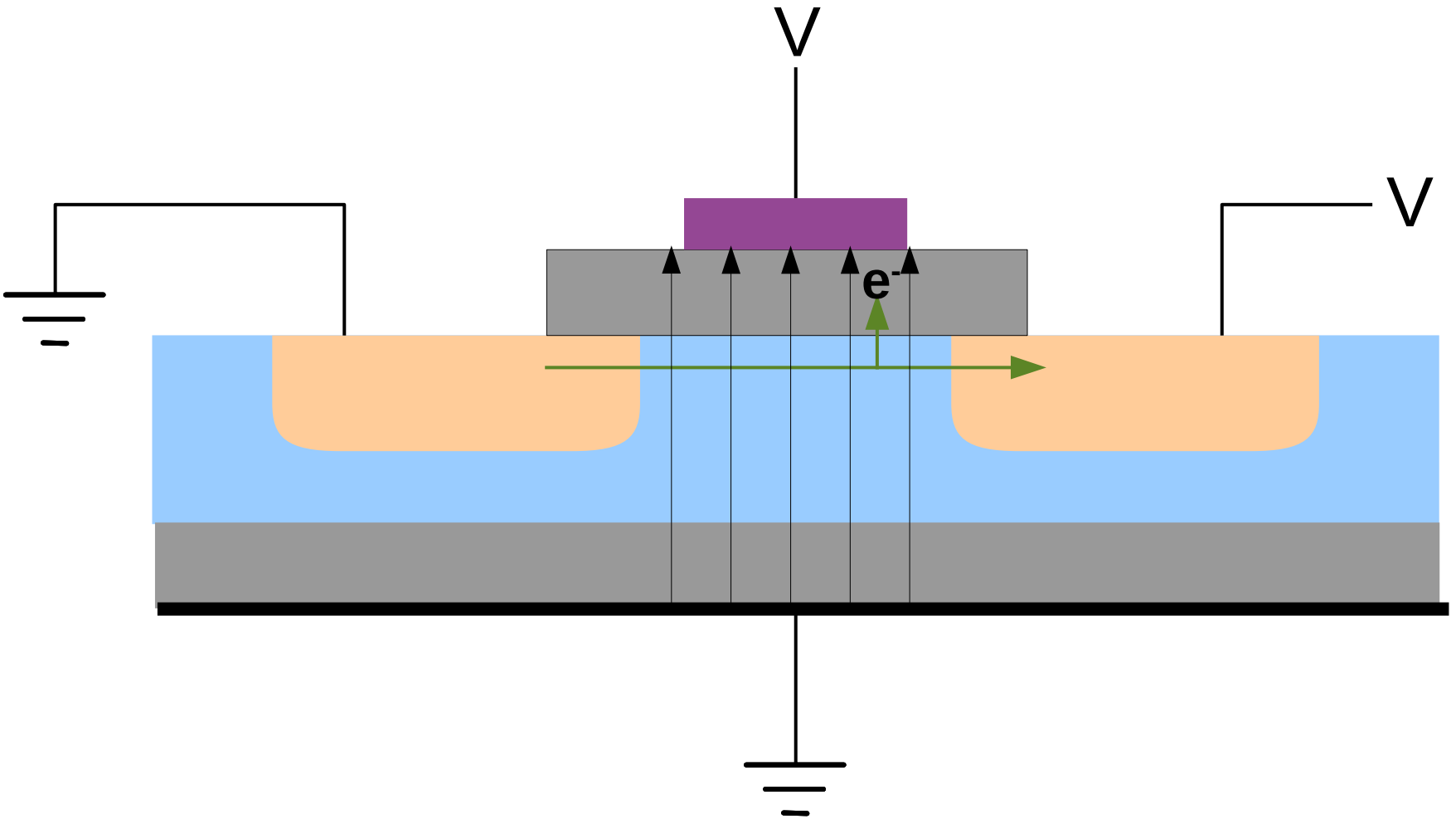
Flash storage technology

Arnout Vandecappelle
arnout@mind.be

http://mind.be/content/Presentation_Flash-technology-ELCE16.odp



Writing the floating gate: Hot carrier injection



Writing the floating gate: Hot carrier injection