CS207 #3, 9 Oct 2009

Gio Wiederhold

http://infolab.stanford.edu/people/gio.html

Gates B12

Correction: Report Drafts for students wanting credit are due 18 November 2009.

Syllabus:

1. Why should software be valued?
2. Open source software. Scope. Theory and reality
3. Principles of valuation. Cost versus value.
4. Market value of software companies.
5. Intellectual capital and property (IP).
6. Life and lag of software innovation.
7. Sales expectations and discounting.
8. The role of patents, copyrights, and trade secrets.
9. Alternate business models.
10. Licensing.
11. Separation of use rights from the property itself.
12. Risks when outsourcing and offshoring development.
13. Effects of using taxhavens to house IP.

Quick definitions: Intangibles

- Software is an intangible good

  If it is owned it is considered Intangible Property

In a business there are 3 parts that have value

(Contribute to potential income)

1. **Tangible goods**: buildings, computers, money
2. The **know-how** of management & employees
3. **Intellectual property**: Software, patents, etc.

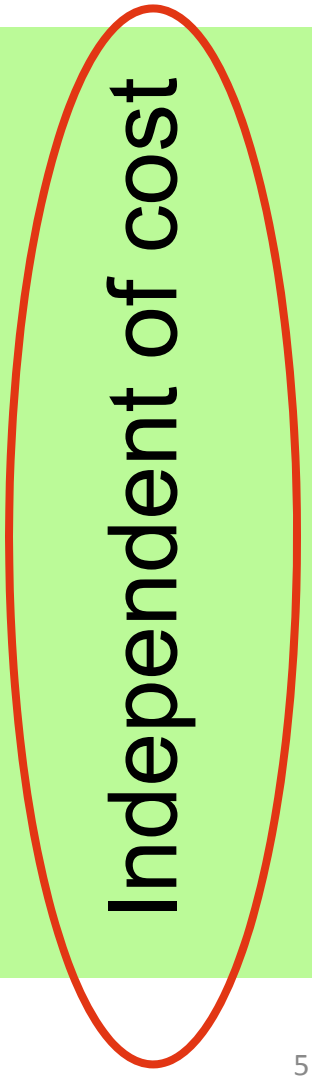
   2. + 3. make up the **Intangible Capital** of a company.

A direct approach

- Value the software specifically by the income it will garner over its lifetime
- But software is not stable over time: *Slithery*
  - Getting long-term income requires maintenance
  - Maintenance enables long-term income
- Much more so than other intangibles
    - Books, music,
- Similar to some intangibles that contribute to life
    - Costumer loyalty, trademarks

Basis for SW value as of today

- Sum of future income
  - Sales = price * copy count
  - Maintenance fees if service subscription
- Minus sum of future costs
  - Cost of goods
  - Cost of marketing
  - Cost of doing business
  - Cost of maintenance
- Discounted to today
  - To account for risk



Technical Parameters needed

IP is to be valued as of some specific date

design, code, ...

1. **Life** of the IP in the product from that time on

   The interval from completion until little of the original *stuff* is left

2. **Diminution of the IP** over the Life

   A bit like a depreciation schedule, but based on content replacement, until little IP is left. 10% is a reasonable limit.

3. **Lag***, interval from transfer to start of IP diminution

   = the time before an investment earns revenue

   - also called "Gestation Period

4. **Relative allocation,** if there are multiple products contributing to income.

Crucial assumption

- **IP content is proportional to SW size**
  - ➢ Not the value, that depends on the income

  ==========

  - ✓ Pro: Programmers efforts create code
    - ✓ An efficient organization will spend money wisely
  - ☒ Counter: not all code contributes equally
    - ✓ early code defines the product, is most valuable
    - ☒ new versions are purchased because of new features
- **Arguments balance out**
  - ✓ it is the best metric we can use

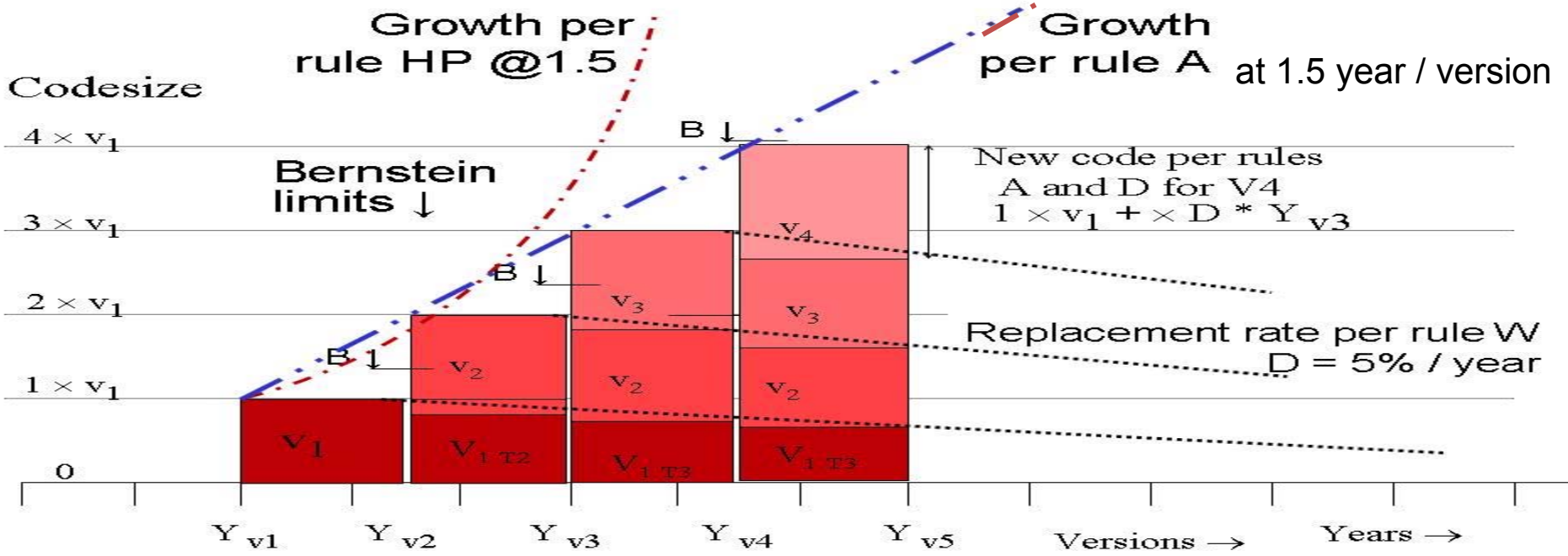
Maintenance → SW Growth

Rules: S_{n+1} = 2 to 1.5 × S_n per year [HennesseyP:90]

$V_{n+1} \leq 1.30\% \times V_n$ [Bernstein:03]

$V_{n+1} = V_n + V_1$ [Roux:97] ([BeladyL72], [Tamai:92,02] indications) [Blum:98]

Deletion of prior code = 5% per year [W:04]



Observations

- Linear growth has been observed, is reasonable
- Software cannot grow exponentially

Because *no Moore's Law*

1. Cost of maintaining software grows exponentially with size
   - The number of interactions among code segments grow faster [Brooks:95]
2. Can't afford to hire staff at exponential $*^2$
3. Cannot have large fraction of changes in a version
   - And get it to be reliable
4. Cannot impose version changes on users < 1 / year
5. Deleting code is risky and of little benefit
   - except in game / embedded code

Price

remember IP = f(income)

- But --- Price stays ≈ fixed over time

  like hardware Moore's Law

  Because

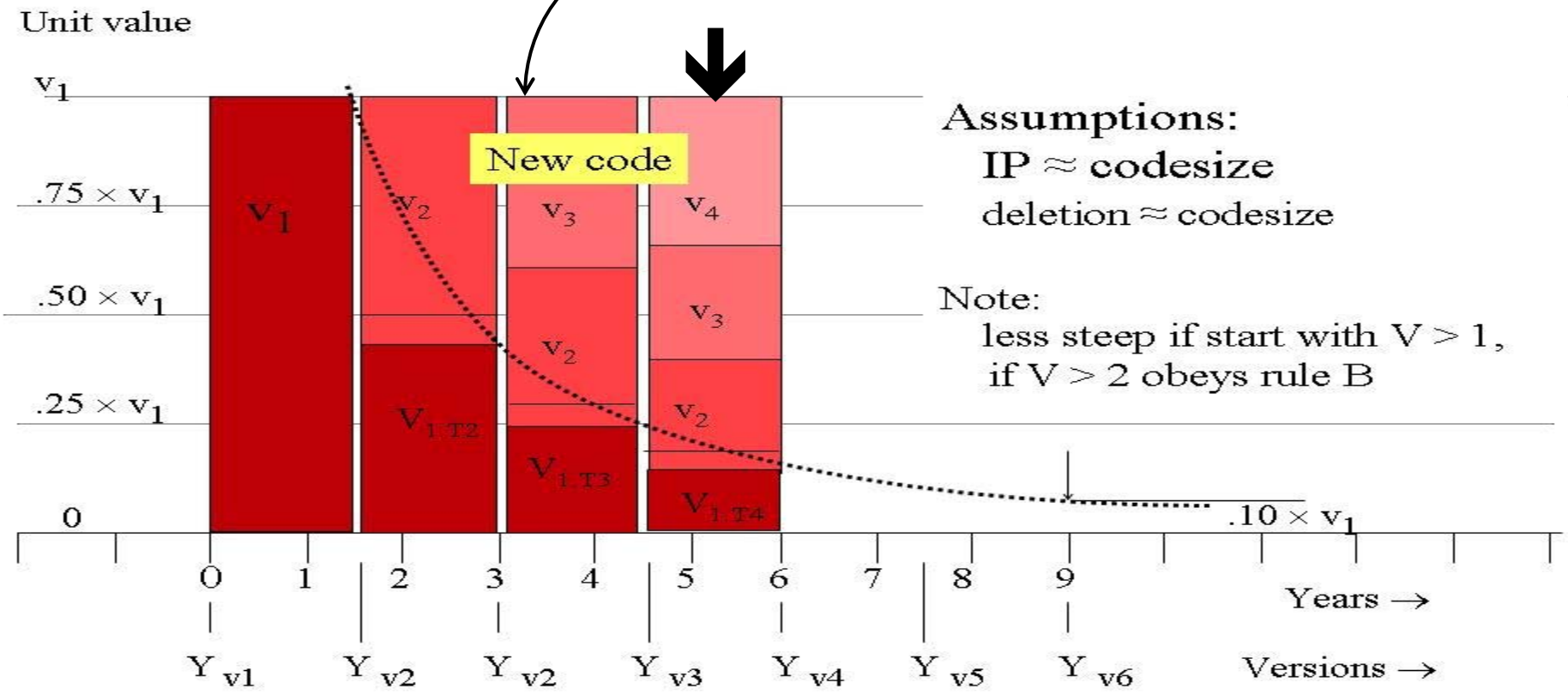
  1. Customers expect to pay same for same functionality
  2. Keep new competitors out
  3. Enterprise contracts are set at 15% of base price
  4. Shrink-wrapped versions can be skipped

- Effect

The income per unit of code reduces by 1 / size →

Growth diminishes IP

For constant unit price



Total income

Total income = price × volume (year of life)

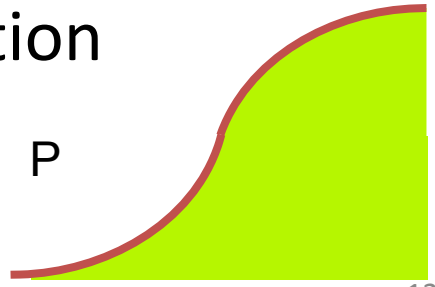
- Hence must estimate volume, lifetime

Best predictors are Previous comparables

- ➢ Erlang curve fitting (m=6 to 20, 12 is typical)

and apply common sense limit = Penetration

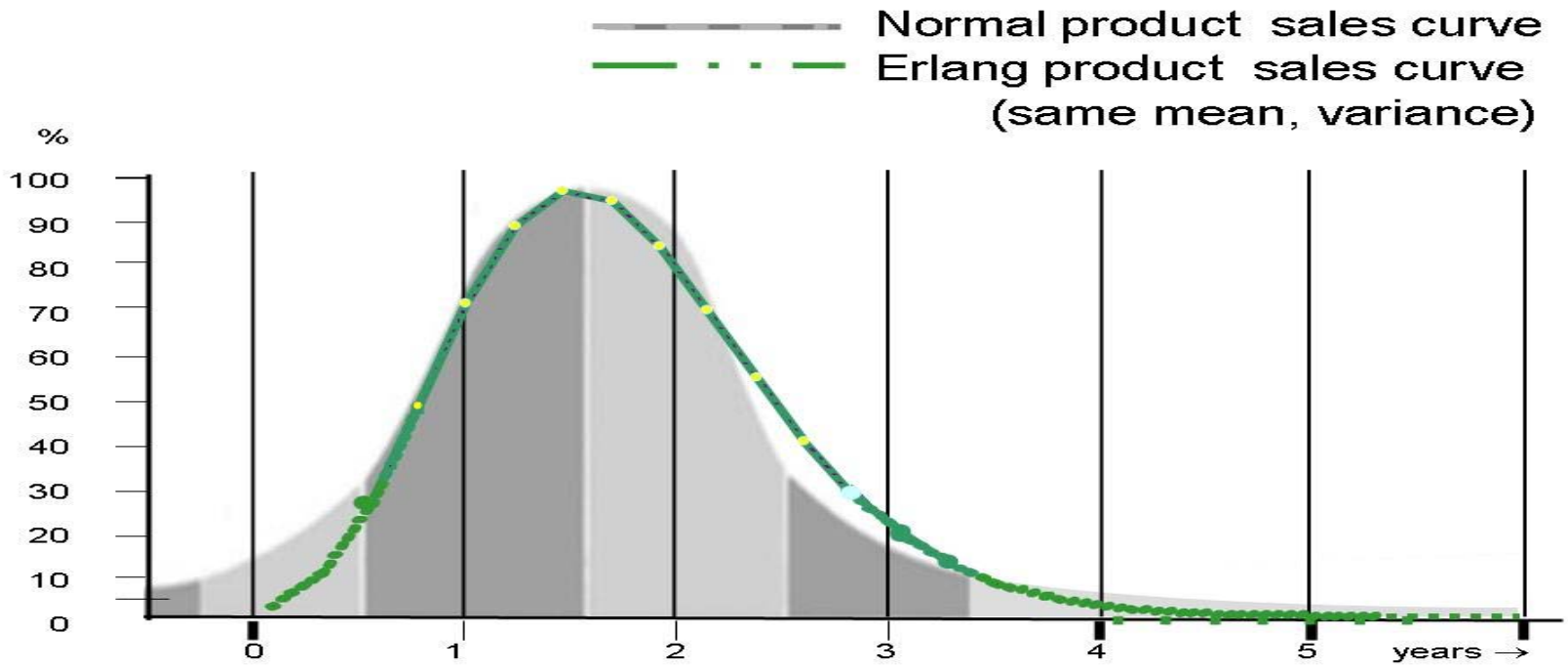
- ➢ estimate total possible sales F × #customers
- ➢ above F= 50% monopolistic aberration



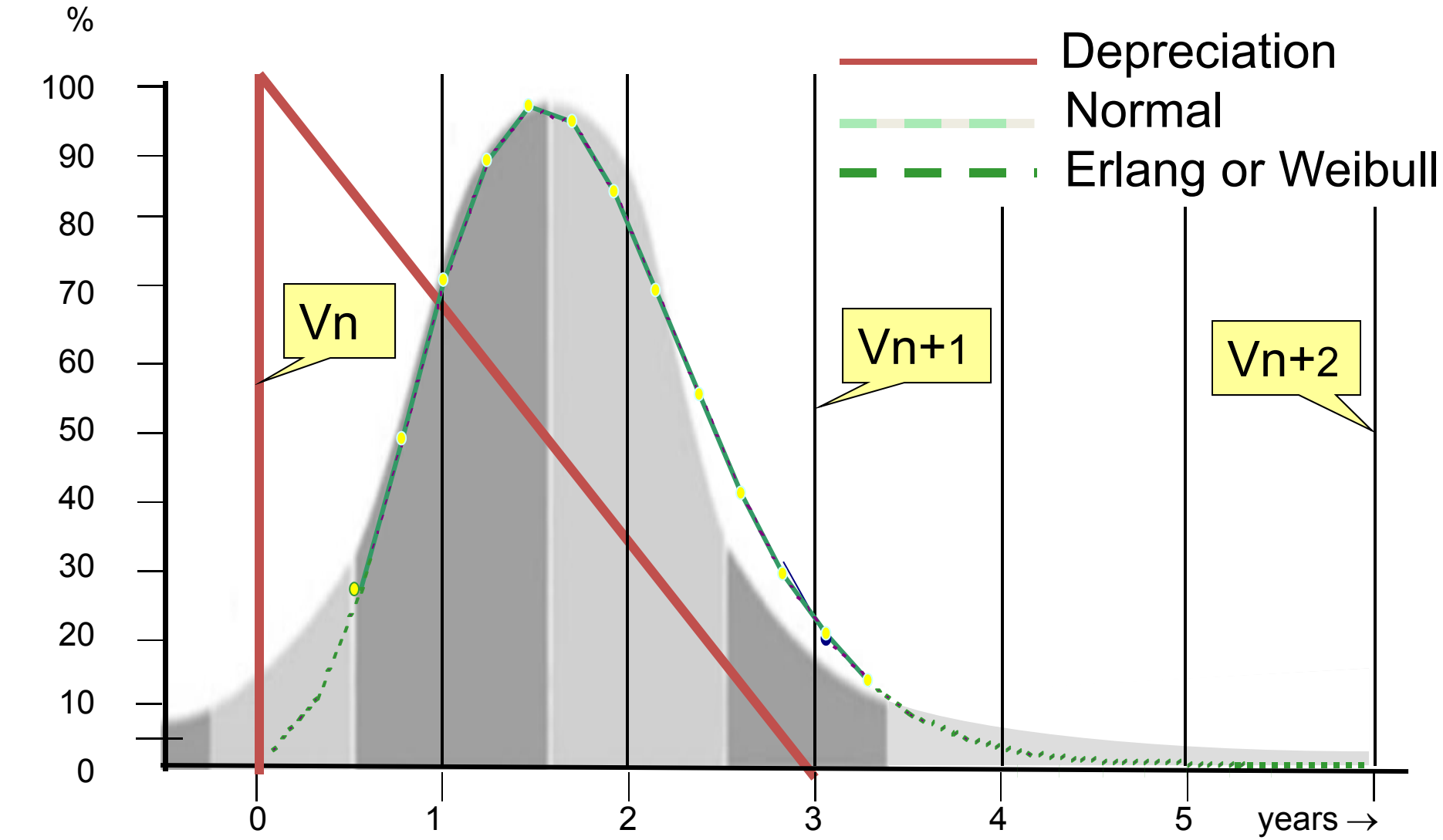
Sales models

1. Normal curve: simple, no defined start point
2. Erlang: realistic, more complex

both have same parameters: mean and variance

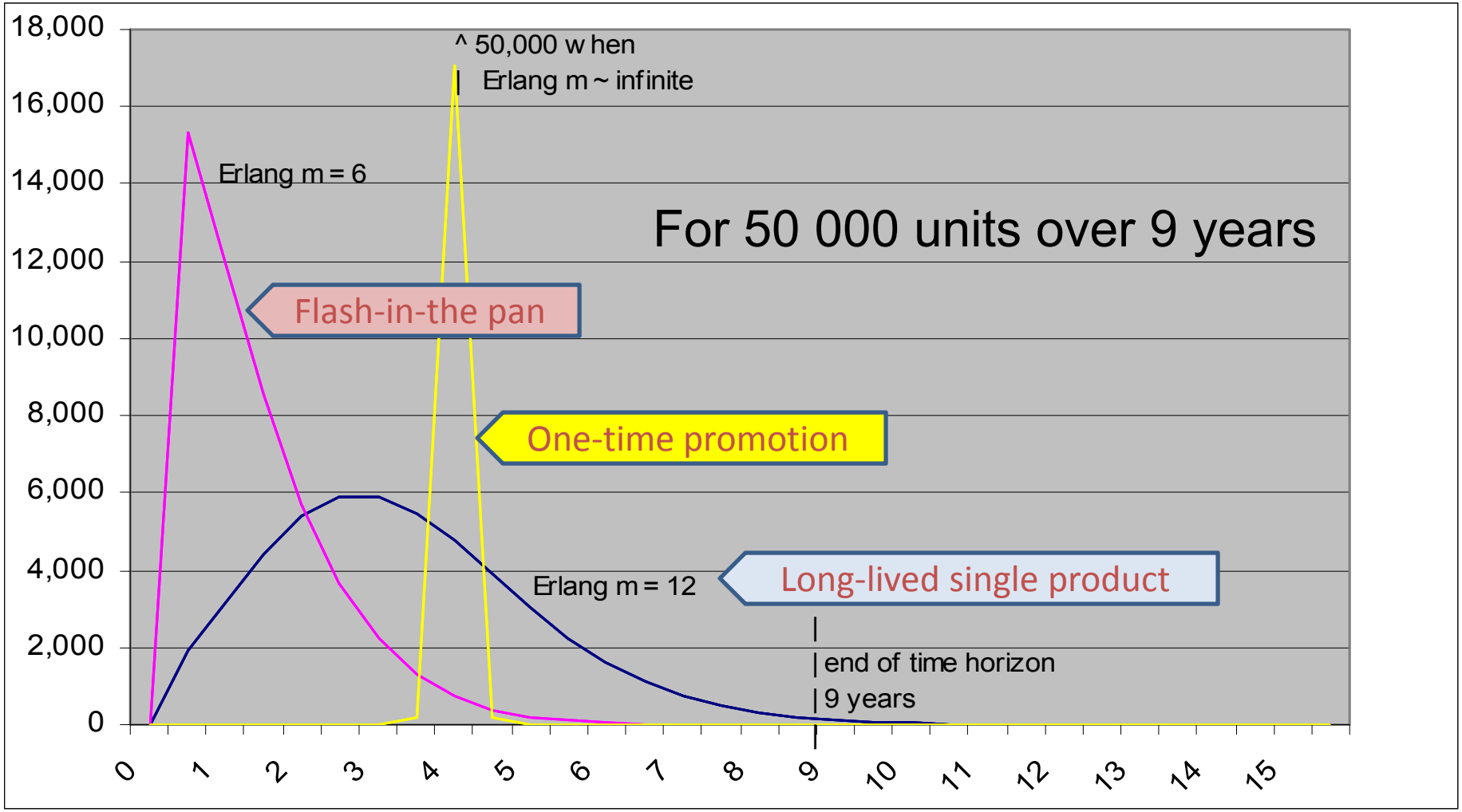


Sales curves



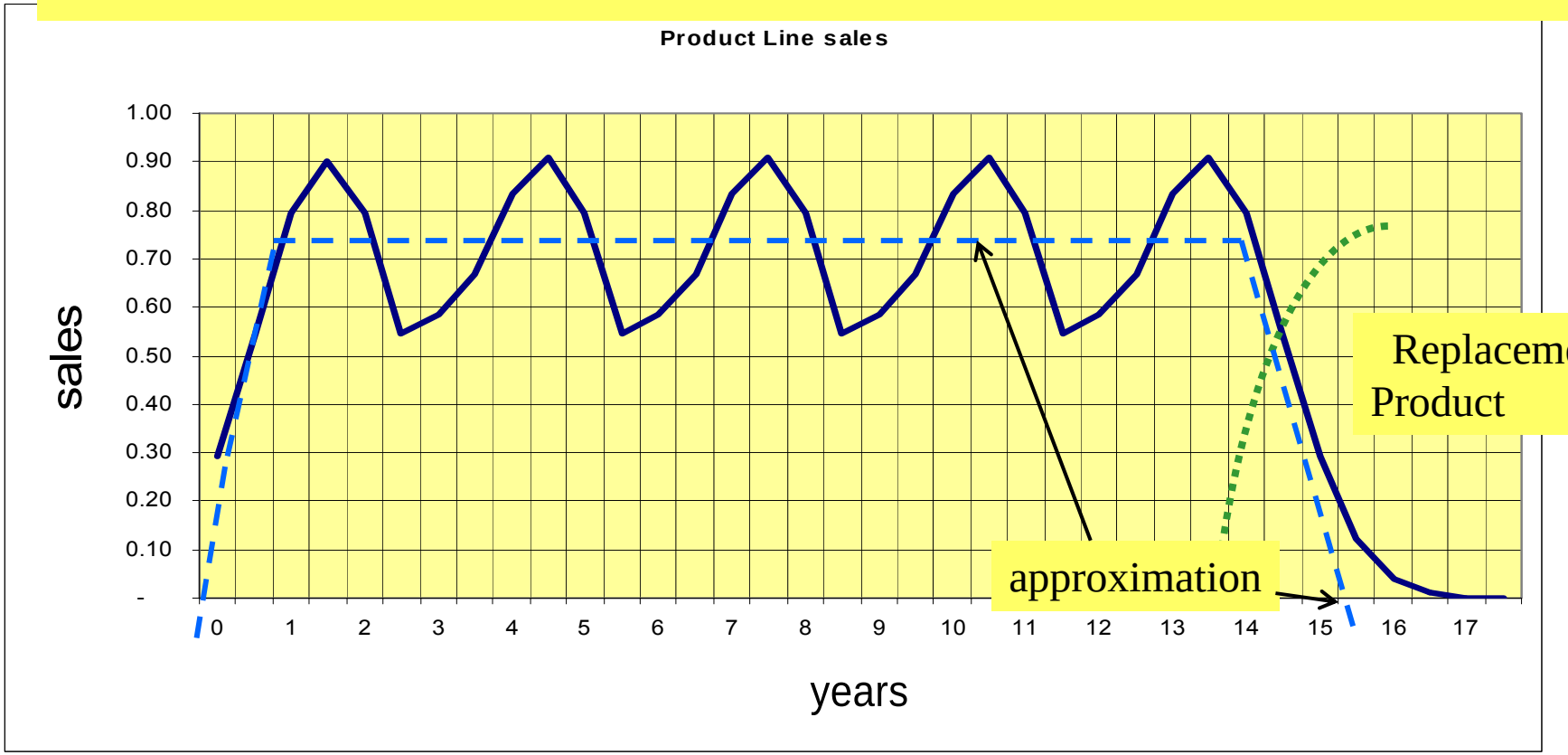
Erlang sales curves

m=mean/variance



Ongoing Version Sales

Predicted product sales for 5 versions, stable rate of product sales
3 year inter-version interval, first-to-last product 12 years, life ~15 years



Fraction of income for SW

Income in a software company is used for

	typical
• Cost of capital	
– Dividends and interest	≈ 5%
• Routine operations -- not requiring IP	
– Distribution, administration, management	≈ 45%
• IP Generating Expenses (IGE)	
– Research and development, i.e., **SW**	**≈ 25%**
– Advertising and marketing	≈ 25%

These numbers are available in annual reports or 10Ks

Discounting to NPV

Standard business procedure

- Net present Value (NPV) of

  getting funds 1 year later = F×(1 – discount %)

Standard values are available for many businesses

  based on risk (β) of business, typical 15%

Discounting strongly reduces effect of the far future

  NPV of $1.- in 9 years at 15% is $0.28

Also means that bad long-term assumptions have less effect

Example

Software product

- Sells for $500/copy
- Market size 200 000
- Market penetration 25%

➢ Expected sales 50 000 units

➢ Expected income $500 x 50 000 = $25M

What is the result?

Combining it all

factor	**today**	y1	y2	y3	y4	y5	y6	y7	y8	y9
Version	1.0	2.0		3.0	4.0		5.0	6.0		7.0
unit price	$500	500	500	500	500	500	500	500	500	500
Rel.size	1.00	1.67	2.33	3.00	3.67	4.33	5.00	5.67	6.33	7.00
New grth	0.00	0.67	1.33	2.00	2.67	3.33	4.00	4.67	5.33	6.00
replaced	0.00	0.05	0.08	0.12	0.15	0.18	0.22	0.25	0.28	0.32
old left	1.00	0.95	0.92	0.88	0.85	0.82	0.78	0.75	0.72	0.68
Fraction	100%	57%	39%	29%	23%	19%	16%	13%	11%	10%
Annual $K	0	1911	7569	11306	11395	8644	2646	1370	1241	503
Rev, $K	0	956	3785	5652	5698	4322	2646	1370	621	252
SWIP25%	0	239	946	1413	1424	1081	661	343	155	63
Due old	0	136	371	416	320	204	104	45	18	6
Disct 15%	1.00	0.87	0.76	0.66	0.57	0.50	0.43	0.38	0.33	0.28
Contribute	0	118	281	274	189	101	45	17	6	2
Total	1032	≈ $ 1 million								

Result of Example

- Selling 50 000 SW units at $500 ≈ $ 1M

  not $ 25M

Once its in a spreadsheet, the effect of the many assumptions made can be checked.

When assumptions later prove unwarranted then management can make corrections.

To be wise, don't spend more than ≈ $500 000 to develop the software product.

Guidance obtained

- We applied an overall Erlang sales curve
  - ➢ new versions keep market going but customers do not replace earlier versions
- The assumption are sufficiently simple that alternatives can be intelligently discussed
  1. keep development costs low
  2. design so that SW maintenance is low
  3. charge a higher price
  4. broaden the market
  5. or →

Business models

0. New versions do not replace earlier versions

Alternative business models

1. New versions encourage replacement
2. Provide related services
3. Charge for maintenance

   Lower initial cost, slower income stream

4. Make product Open source to broaden market

   Charge only for services

Discussion

- Many choices now

a. Technical

b. Business

Interact with each other.